Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004:007
DOI: https://doi.org/10.24867/29OI05Melovic

SISTEMATSKI PREGLED TEHNIKA I ALATA ZA OPTIMIZACIJU KORIŠĆENJA SWIFT PROGRAMSKOG JEZIKA

SYSTEMATIC REVIEW OF TECHNIQUES AND TOOLS FOR OPTIMIZING USAGE OF THE SWIFT PROGRAMMING LANGUAGE

Sandra Melović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – INFORMACIONO-KOMUNIKACIONI SISTEMI

Kratak sadržaj – *Ovaj rad pruža sistematski pregled literature o tehnikama i alatima koje programeri koriste za prevazilaženje izazova sa kojima se suočavaju pri korišćenju Swift programskog jezika. Rad istražuje različite kategorije problema koji utiču na performantnost aplikacija pisanih u Swift-u. Ističe prednosti i ograničenja trenutnih pristupa što omogućava programerima da efikasnije koriste Swift i poboljšaju performanse svojih aplikacija. Takođe, sugeriše pravce za buduća istraživanja koja bi mogla doprineti daljem unapređenju praksi i alata za rad sa pomenutim jezikom.*

Ključne reči: *Swift, Programski jezik, iOS, Objective-C, Evaluacija, Performanse, Kvalitet koda*

Abstract – *This paper provides a systematic literature review of the techniques and tools that developers use to overcome the challenges they face when using the Swift programming language. The paper investigates different categories of problems that affect the performance of applications written in Swift. It highlights the strengths and limitations of current approaches that allow developers to use Swift more effectively and improve the performance of their applications. It also suggests directions for future research that could contribute to the further improvement of practices and tools for working with the mentioned language.*

Keywords: *Swift, Programming language, iOS, Objective-C, Evaluation, Performance, Code quality*

1. UVOD

Swift je savremen programski jezik, razvijen od strane *Apple*-a, u cilju unapređenja razvoja aplikacija za iOS, macOS, watchOS, tvOS, *Linux* i z/OS platforme. Činjenica da je ušao u 10 najkorišćenijih jezika i za izuzetno kratko vreme zamenio pređašnje korišćeni *Objective-C* pokazuje da je jedan od najbrže rastućih jezika. Popularnost je stekao kako zbog svoje jednostavnosti, stabilnosti i robusnosti, tako i zbog *open-source* osobine, koja omogućava njegovo besplatno korišćenje i modifikaciju. *Swift*, iako moćan, nije bez izazova i nedostataka, što ga čini zanimljivim predmetom istraživanja u oblasti optimizacije performansi.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bila dr Dušanka Dakić, docent.

Cilj ovog rada je sistematski pregled literature radi identifikacije izazova sa kojima se susreću programeri prilikom korišćenja *Swift* programskog jezika, kao i pronalaženje efikasnih metoda za prevazilaženje identifikovanih problema. Analiza postojećih istraživanja pružiće uvid u pregled tehnika i alata za poboljšanje performansi aplikacija razvijenih u ovom jeziku.

U drugom poglavlju će biti prikazan istorijat i razvoj *Swift* progrmaskog jezika. Treće poglavlje predstavlja metodologiju sistemskog pregleda literature, dok se u četvrtom nalaze rezultati i diskusija, u kojoj će biti reči o trenutnoj praksi i identifikovanim problemima, kao i primeni optimizacijskih tehnika i konkretnih poboljšanja pri korišćenju *Swift*-a u praksi. Na kraju, u odeljku broj pet, dat je zaključak.

2. ISTORIJAT I RAZVOJ SWIFT-A

Chris Lattner 2010. godine započinje rad na *Swift*-u, koji je u tom trenutku bio tajni projekat unutar *Apple*-a. Sa krajnjim ciljem da se zameni *Objective-C*, ovaj relativno mlad jezik, predstavljen je 2014. godine na *Apple*-ovoj WWDC (*The Worldwide Developers Conference*) konferenciji. Od svog prvog pojavljivanja, *Swift* je doživeo značajne promene. Druga verzija pojavljuje se već naredne godine, donoseći značajna poboljšanja u pogledu funkcionalnosti i performantnosti. Iste godine *Swift* postaje *open-source*, što ga čini još popularnijim i omogućava zajednici da doprinosi razvoju jezika. Godine 2016. objavljen je *Swift 3*, koji predstavlja jednu od najvećih i najznačajnijih nadogradnji, sa fokusom na povećanje nivoa stabilnosti i performanse. Tranzicija sa druge na treću verziju je bila toliko teška programerima da su neki projekti pisani od nule. U narednim verzijama, jezgro se nije drastično menjalo, stoga su četvrta i peta verzija kompatibilne sa trećom. U okviru njih, radilo se na poboljšanjima implementacije pojedinih koncepata jezika. Poslednja verzija, 5.10.1, izbačena je u junu 2024.

2. METODOLOGIJA

Fokus sistematskog pregleda literature je da pruži sveobuhvatan rezime postojećih istraživanja o određenoj temi. Za ovaj pregled korišćen je PRISMA (*Preferred Reporting Items for Systematic Reviews and Meta-Analyses*) protokol. On uključuje definisanje ključnih tema istraživanja, selekciju relevantnih izvora publikacija, odabir odgovarajućih ključnih reči za pretragu, pretraživanje relevantne literature, kao i identifikaciju

mogućih praznina u istraživanjima radi jačanja polja istraživanja.

2.1. Indeksne baze

Za ovaj sistematski pregled literature kao indeksne baze korišćeni su *Scopus* i *Google Scholar*. Kroz sistemsku

pretragu *Scopus* baze, identifikovani su ključni radovi koji su doprineli razumevanju i analizi date teme. Za *Google Scholar* je korišćen upit „iOS *Swift reengineering*“. *Scopus* je pretražen kroz: „*Swift*“ AND „iOS“. Upit za *Scopus* je opštijeg tipa, zbog veoma malog broja radova koji se dobiju prilikom preciznijeg definisanja upita.

Kombinacija *Scopus*-a i *Google Scholar*-a omogućila je sveobuhvatno istraživanje relevantnih radova. *Scopus* je poslužio kao ključna indeksna baza za detaljnu analizu citata, kvaliteta i konteksta istraživanja, dok je *Google Scholar* bio koristan za inicijalno prepoznavanje radova i šireg pregleda literature.

2.2. Istraživačka pitanja

U skladu s ciljem istraživanja postavljaju se sledeća istraživačka pitanja:

IP1: Kako programeri percipiraju *Swift* u poređenju sa drugim programskim jezicima?

IP2: Koji su najčešći problemi i izazovi sa kojima se programeri suočavaju prilikom korišćenja *Swift* jezika?

IP3: Koje su najefikasnije tehnike i alati za optimizaciju performansi *Swift* aplikacija?

2.3. Kriterijumi inkluzije i ekskluzije

Kako bi se osiguralo da su radovi uključeni u istraživanje značajni za evaluaciju i poboljšanje korišćenja *Swift* programskog jezika korišćeni su dole navedeni kriterijumi prihvatanja.

Kriterijumi inkluzije:

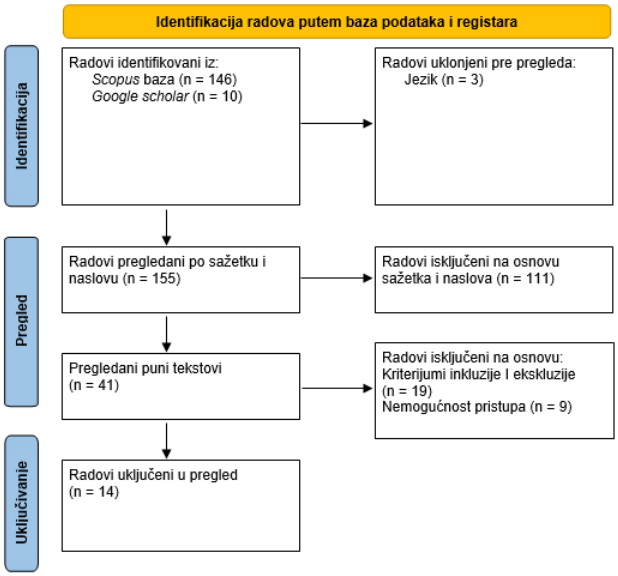
1. Istraživanja koja se fokusiraju na *Swift* programski jezik, njegove karakteristike, primene i performanse.
2. Radovi koji se bave tehnikama i metodologijama za optimizaciju performansi koda pisanog u *Swift*-u.
3. Istraživanja koja analiziraju korisničko iskustvo, razvojne procese i prakse u korišćenju *Swift*-a.
4. Istraživanja koja upoređuju *Swift* sa drugim programskim jezicima u kontekstu performansi i korišćenja.
5. Istraživanja koja su na engleskom jeziku.

Kriterijumi ekskluzije:

1. Istraživanja koji pominju *Swift* usputno, bez dublje analize ili evaluacije.
2. Istraživanja koja ne pružaju empirijske dokaze ili praktične primere.

2.4. PRISMA protokol

Na slici 1 predstavljen je proces filtriranja radova po PRISMA protokolu. Pretragom gore navedenih indeksnih baza identifikovano je 156 studija. Filtriranjem po jeziku, gde su ostavljene samo studije koje su napisane na engleskom jeziku, isključene su 3 studije. Nakon toga pregledani su naslovi i sažeci čime je isključeno 111 očigledno nerelevantnih studija. Preostale studije su detaljno procenjene čitanjem punih tekstova kako bi se utvrdila njihova podobnost za uključivanje u pregled. Primenom kriterijuma prihvatanja eliminisano je 19 radova. Na samom kraju zbog nemogućnosti pristupa, iz pregleda je izopšteno još 9 studija, čime se konačna selekcija završava na 14 radova.



Slika 1. *Proces filtriranja radova po PRISMA protokolu*

3. REZULTATI I DISKUSIJA

IP1 - *Kako programeri percipiraju Swift u poređenju sa drugim programskim jezicima?* *Swift*, kao savremen i efikasan jezik koji se stalno unapređuje, te nudi visoku produktivnost i sigurnost, u očima programera stekao je veliku popularnost. Percipiran je lakšim za učenje i korišćenje od *Objective-C*. Istraživanjem „On the Usage of Programming Languages in the iOS Ecosystem“ je pokazano da 73% aplikacija koristi *Swift*, 21% aplikacija koristi *Objective-C*, dok 6% koristi C kao glavni programski jezik [1]. Uvođenjem novih funkcionalnosti poput *Switch* strukture, *Generics* i *Closures*, *Swift* omogućava lakše čitanje i pisanje koda, što rezultira manjim brojem grešaka [2]. U određenim testovima brži je od *Java*-e, C++ i C#, posebno u zadacima kao što su *Fibonacci* sekvenciranje i rad sa kolekcijama, iako nije uvek značajno brži u svim programima [3].

U poređenju sa *Kotlin*-om, *Swift* ga nadmašuje u pogledu vremena izvršavanja, potrošnje memorije i upotrebe CPU (The Central Processing Unit). Međutim, velika prednost *Kotlin*-a predstavlja deljenje koda između *Android* i iOS platformi, dok se *Swift* koristi tradicionalno za razvoj isključivo iOS aplikacija [4]. Istraživanja su pokazala da multiplatformski *React Native* takođe može pružiti bolje performanse od *Swift*-a u određenim scenarijima, ali kada se radi o renderovanju većeg broja elemenata na ekranu, *Swift* pokazuje značajno bolje performanse i to uz manju potrošnju memorije i CPU (*The Central Processing Unit*) resursa. Još jedna prednost *Swift*-a je rad sa *nullable* tipovima podataka koji smanjuju rizik od pojavljivanja

Null Pointer Exceptions [5]. Uprkos izazovima, programeri prepoznaju *Swift* kao moćan alat za kreiranje GUI (*Graphical User Interface*) baziranog softvera, što ga čini atraktivnim izborom za razvoj na *Apple* platformama.

IP2 - *Koji su najčešći problemi i izazovi sa kojima se programeri suočavaju prilikom korišćenja Swift jezika?* Iako, po mišljenju programera, prilično jednostavan za upotrebu, *Swift* programski jezik nije poštedeo svoje korisnike problema i izazova pri korišćenju. Programeri se suočavaju sa problemima vezanim za ranjivost paketa, kao i nedostatak testne pokrivenosti (koja iznosi svega 1%) [6], što može otežati održavanje i proširivanje poslovne logike aplikacija. Iako jednostavne sintakse, rad sa nizovima, tipovima podataka *optionals* i obradom grešaka programeri percipiraju komplikovanim, specifično za one koji prelaze sa *Objective-C*, s obzirom na to da prilagođavanje novim konceptima može zahtevati vreme i dodatno obrazovanje [7]. Česta su i pitanja vezana za skladištenje podataka i višenitne probleme, kao i *bugs* i *code smells* koji se neujednačeno javljaju u komponentama i izvornim datotekama, a njihovi simptomi se manifestuju na različite načine, uglavnom u obliku pada softvera [8]. Jedan od ključnih problema koji utiče na njihov rad i efikasnost je i ranjivost u paketima, pri čemu je otkriveno da *Swift* ekosistem ima manje javnih ranjivosti u poređenju sa drugim jezicima, ali i dalje nedostaju alati za detaljnu analizu ranjivih zavisnosti [9]. Iz navedenog se može zaključiti očigledan izostanak alata koji, između ostalog, mogu efikasno analizirati kod radi osiguranja kvaliteta i detekcije tehničkog duga u ranoj fazi razvoja. Tabelom 1 dat je prikaz detektovanih problema kroz domene.

IP3 - *Koje su najefikasnije tehnike i alati za optimizaciju performansi Swift aplikacija?* Kombinovanjem različitih tehnika i alata, programeri mogu razvijati aplikacije koje, pored efikasnosti, odlikuje održivost, pouzdanost i performantnost, omogućavajući, kako lak razvoj, tako i bolje korisničko iskustvo. Alati za analizu koda kao što su *SwiftLangDoc 2019* i *UberRibs 2017* pružaju korisne smernice za efikasniju upotrebu i organizaciju *Swift* koda [10]. Takođe, alati kao što su *SonarSource* i *CodeClimate*, *Paprika* i *GraphifySwift* pružaju statičku analizu koda, otkrivajući različite tipove ranjivosti, poput *bug*-ova i *code smells* [11]. SCMA (*Swift Code Metrics Analyzer*) i *SwiftLint* proveravaju, respektivno, 10 i 12 metrika koda, što pomaže programerima da optimizuju performanse svojih aplikacija i osiguraju kvalitet koda [12]. *iPerfDetector* je alat koji se koristi za detekciju anti-uzoraka performansi u iOS aplikacijama napisanih u *Swift*-u. Ovaj alat uspešno prati upotrebu i pozive objektnih istanci, detektujući anti-uzorke koji negativno utiču na performanse. Istraživanje „iPerfDetector: Characterizing and detecting performance anti-patterns in iOS applications“ pokazuje da je u proučenim aplikacijama detektovao 34 instance anti-uzoraka, od kojih su 32 ručno verifikovane [13]. Korišćenje *Instruments* alata za analizu performansi pomaže u identifikaciji i rešavanju problema sa memorijom. Optimizacija protokola kroz tri kompajlerske optimizacije, kao i upravljanje zavisnostima putem *Swift Package Managera*, integrisanim u *X-Code* IDE (*Integrated development environment*), potpomaže performantnijem razvoju. Tehnike poput *lazy loading* i *async*/*await* implementacije, kao i prelazak na MVC (*Model-View-Controller*) arhitekturu, mogu značajno poboljšati performanse aplikacija [5].

Tabela 1. *Detektovani problemi kroz kategorije*

Kategorije	Aspekti	Radovi
Kvalitet koda	Nedostatak alata za analizu	(Rahkema & Pfahl, 2023)
	Detekcija anti-uzoraka performansi	(Afjehei & Chen & Tsantalis, 2019)
	Bugs	(Mai, 2023)
	Code smells	(Rahkema & Pfahl, 2020), (Rabbi & Hossain & Arefin, 2022), (Afjehei & Chen & Tsantalis, 2019)
Sigurnost	Ranjivosti u zavisnostima i bibliotekama	(Rahkema & Pfahl, 2023)
Održavanje	Nedostatak testne pokrivenosti	(Stulz, 2020)
Prevazilaženje jezičkih razlika	Prelaz sa *Objective-C* na *Swift*, razlike u sintaksi i pristupima	(Reboucas & Pinto & Ebert & Torres & Serebrenik & Castor, 2016)
Specifični jezički izazovi	Rad sa *optionals*	(Reboucas & Pinto & Ebert & Torres & Serebrenik & Castor, 2016)
	Obrada grešaka	(Reboucas & Pinto & Ebert & Torres & Serebrenik & Castor, 2016)
	Nizovi	(Reboucas & Pinto & Ebert & Torres & Serebrenik & Castor, 2016)
	Tipovi podataka	(Reboucas & Pinto & Ebert & Torres & Serebrenik & Castor, 2016)
Edukacija	Nedostatak dokumentacije, resursa za učenje i podrške	(Chakraborty & Shahriyar & Iqbal & Uddin, 2021)

Pored navedenih tehnika, korišćenje najnovijih verzija *X-Code*-a, pravilno korišćenje *Swift* biblioteka i SDK-ova (*Software Development Kit*), posebno *Foundation Kit*-a, kao i praćenje diskusija na platformama kao što je *Stack Overflow*, dodatno može olakšati pisanje koda i doprineti optimizaciji performansi [14]. Tabela 2 prikazuje alate i tehnike kroz domene.

4. ZAKLJUČAK

Analiza postojećih istraživanja koji se bave tehnikama i alatima za optimizaciju performansi pruža sistematski pregled metoda koje programeri mogu koristiti za efikasniju upotrebu *Swift* programskog jezika. Korisna je jer pokriva širok spektar pristupa, koji se temelje na empirijskim podacima, što potvrđuje efikasnost predloženih rešenja. Specifičnost prema domenima problema olakšava identifikaciju i rešavanje konkretnih izazova, dok raznovrsnost alata i tehnika omogućava prilagodljivost različitim potrebama programera. Naravno, analiza ima i svoja ograničenja koja se ogleda u pokrivenosti svih specifičnih problema koji se mogu javiti. Pristup alatima, takođe, može biti ograničavajući faktor za timove kojima je, iz određenog razloga (zahtev za posebnim licencama, kompatibilnost sa određenim operativnim sistemima, zauzimanje značajnih računarskih resursa itd.) onemogućen. Postavlja se i pitanje održivosti optimizacija, s obzirom na konstantne modifikacije jezika i mogućnost zastarevanja. Upravo iz ovih razloga, dalje ideje istraživanja mogu biti sprovođenje studija za procenu dugoročnog uticaja optimizacija, kao i razvoj novih alata prilagođenih specifičnim potrebama *Swift* programera. Još jedan od pravaca, koji može biti itekako inspirativan, jeste primena mašinskog učenja za

automatsku detekciju problema. Kontinuirano praćenje promena jezika i adaptacija na iste ključna je za iskorištavanje svih prednosti jezika. Ovakvim pristupom, programerima će biti omogućeno da iskoriste pun potencijal jezika, kreirajući visokokvalitetne softvere koji zadovoljavaju narastajuće zahteve modernog tržišta.

Tabela 2. *Tehnike i alati kroz kategorije problema*

Kategorija	Tehnika/Alat	Opis	Radovi
Kvalitet koda	Optimizacija *Swift* protokola	Ubrzanja izvršavanja od 6.9% do 55.49%.	(Barik & Sridharan & Ramanathan & Chabbi, 2019)
	Instruments	Analiza performansi i rešavanje problema sa memorijom.	(Bilberg, 2018)
	iPerfDetector	Detektovanje anti-uzoraka performansi.	(Afjehei & Chen & Tsantalis, 2019)
	SCMA	Analiza deset metrika koda za poboljšanje arhitekture softvera.	(Rabbi & Hossain & Arefin, 2022)
	SonarSource	Statička analiza koda.	(Rahkema & Pfahl, 2020)
	Code Climate	Statička analiza koda.	(Rahkema & Pfahl, 2020)
	SwiftLint	Analiza preko 200 pravila, uključujući 12 metrika koda.	(Rahkema & Pfahl, 2020)
	Paprika	Analiza *code smell*-ova.	(Rahkema & Pfahl, 2020)
	GraphifySwift	Analiza *code smell*-ova.	(Rahkema & Pfahl, 2020)
Sigurnost	Ažuriranje direktnih zavisnosti	Održavanje sigurnosti i performansi.	(Rahkema & Pfahl, 2023)
Održavanje	*Swift Package Manager*	Olakšavanje procesa specifikovanja novih modula i korišćenja koda sa lokalnih i udaljenih izvora.	(Bilberg, 2018)
	"Edit and Pray" i *"Cover and Modify"*	Strategije za pisanje testova.	(Stulz, 2020)
Prevazilaženje jezičkih razlika	Upravljanje bibliotekama	Pravilno korišćenje *Swift* biblioteka i SDK-ova, posebno *Foundation Kit*-a.	(Chakraborty & Shahriyar & Iqbal & Uddin, 2021)
	Praćenje verzija *X-Code*-a	Korišćenje najnovijih verzija za bolje performanse.	(Chakraborty & Shahriyar & Iqbal & Uddin, 2021)
Specifični jezički izazovi	*Tuples*, *Generics*, *Closures*, *Switch*	Implementacija ovih tehnika za poboljšanje performansi.	(García, Espada, G-Bustelo, Lovelle, 2015), (Fojtik, 2020)
	Prilagođavanje arhitekture	Prelazak na MVC strukturu za lakše upravljanje elementima.	(Bilberg, 2018)
Edukacija	*SwiftLangDoc 2019*	Bolja upotreba i organizacija koda.	(Barik & Sridharan & Ramanathan & Chabbi, 2019)
	UberRibs 2017	Bolja upotreba i organizacija koda.	(Barik & Sridharan & Ramanathan & Chabbi, 2019)
	Stack Overflow	Uočavanje novina i boljih praksi.	(Chakraborty & Shahriyar & Iqbal & Uddin, 2021)

5. LITERATURA

[1] D. Dominguez-Alvarez, A. Gorla, and J. Caballero, 'On the Usage of Programming Languages in the iOS Ecosystem', in *2022 IEEE 22nd International Working Conference on Source Code Analysis and Manipulation (SCAM)*, Limassol, Cyprus: IEEE, Oct. 2022, pp. 176–180.

[2] C. González García, J. Pascual-Espada, C. Pelayo G-Bustelo, and J. M. Cueva-Lovelle, 'Swift vs. Objective-C: A New Programming Language', *IJIMAI*, vol. 3, no. 3, p. 74, 2015.

[3] R. Fojtik, 'Swift a New Programming Language for Development and Education', in *Digital Science 2019*, vol. 1114, T. Antipova and Á. Rocha, Eds., in Advances in Intelligent Systems and Computing, vol. 1114. Cham: Springer International Publishing, 2020, pp. 284–295.

[4] https://kth.diva-portal.org/smash/get/diva2:1793389/FULLTEXT01.pdf (pristupljeno u junu 2024.)

[5] https://www.diva-portal.org/smash/get/diva2:1215717/FULLTEXT01.pdf (pristupljeno u junu 2024.)

[6] https://scg.unibe.ch/archive/projects/Stul20a.pdf (pristupljeno u junu 2024.)

[7] M. Reboucas, G. Pinto, F. Ebert, W. Torres, A. Serebrenik, and F. Castor, 'An Empirical Study on the Usage of the Swift Programming Language', in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Suita: IEEE, Mar. 2016, pp. 634–638.

[8] 'Toward Understanding Bugs in Swift Programming Language', in *2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS)*, Chiang Mai, Thailand: IEEE, 2023.

[9] K. Rahkema and D. Pfahl, 'Vulnerability Propagation in Package Managers Used in iOS Development', in *2023 IEEE/ACM 10th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, May 2023, pp. 60–69.

[10] R. Barik, M. Sridharan, M. K. Ramanathan, and M. Chabbi, 'Optimization of swift protocols', *Proc. ACM Program. Lang.*, vol. 3, no. OOPSLA, pp. 1–27, Oct. 2019.

[11] K. Rahkema and D. Pfahl, 'Comparison of Code Smells in iOS and Android Applications', 2020.

[12] F. Rabbi, S. S. Hossain, and M. M. S. Arefin, 'SCMA: A Lightweight Tool to Analyze Swift Projects', presented at the The 34th International Conference on Software Engineering and Knowledge Engineering, Jul. 2022, pp. 440–443.

[13] S. S. Afjehei, T.-H. (Peter) Chen, and N. Tsantalis, 'iPerfDetector: Characterizing and detecting performance anti-patterns in iOS applications', *Empir Software Eng*, vol. 24, no. 6, pp. 3484–3513, Dec. 2019.

[14] P. Chakraborty, R. Shahriyar, A. Iqbal, and G. Uddin, 'How do developers discuss and support new programming languages in technical Q&A site? An empirical study of Go, Swift, and Rust in Stack Overflow', *Information and Software Technology*, vol. 137, p. 106603, Sep. 2021.

Kratka biografija:

Sandra Melović rođena je u Prijepolju 2000. god. 13. septembra 2023. godine, završava osnovne akademske studije i stiče zvanje diplomiranog inženjera informacionih tehnologija, sa temom diplomskog rada, "Implementacija informacionog sistema za podršku poslovanju objedinjenog sistema auto škole", pod mentorstvom doc. dr Teodore Lolić.
kontakt: sandramelovic00 @gmail.com