UDC: 316.774:004.4 Info M: str. 27-35

ANALIZA AKADEMSKOG PLAGIRANJA IZVORNOG KODA
ANALYSIS OF ACADEMIC SOURCE CODE PLAGIARISM

Dejan Ćapara

REZIME: U današnjem informacionom svetu, računarski programi se u nedovoljnoj meri čuvaju od malverzacija i neautorizovanog korišćenja. Značajan oblik kršenja autorskih prava predstavlja plagijarizam. Veoma je zastupljen u programiranju, naročito na primeru izvornog koda, koji je najosetljivija meta plagiranja. Kao i plagijarizmi drugih formi intelektualne svojine, najzastupljeniji je u akademskoj sredini. Studentski radovi su izloženi različitim malverzacijama, a profesorima su od velike koristi u sprečavanju ovakvih prestupa različiti alati za detekciju plagijarizma. U radu je data komparativna analiza alata odnosno algoritama koji se koriste za prevenciju plagiranja izvornog koda u akademskom okruženju. Dat je kratak osvrt na pravne norme koje se odnose na računarske programe kao autorska dela. Detaljno su opisana tri alata koji se najčešće primenjuju za detekciju akademskog plagijarizma - JPlag, MOSS i SIM. Najznačajniji rezultati komparacije su prikazani, istaknute su prednosti i mane svakog od alata, mogućnosti primene u različitim situacijama, kao i predlog daljeg razvoja ovih alata.

KLJUČNE REČI: Plagijarizam, akademsko plagiranje, izvorni kod, detekcija plagijarizma, alati, komparativna analiza

ABSTRACT: In today's world of Informatics, computer programs are insufficiently protected against fraud and unauthorized use. A significant form of copyright infringement represents plagiarism. It is very present in programming, particularly in the case of the source code, which is the most sensitive target of plagiarism. As well as other forms of plagiarism of intellectual property, it's mostly present in the academic environment. Student works are exposed to fraud, and to prevent such offenses teachers may use different tools to detect plagiarism. The subject of this research is a comparative analysis of the tools used for the prevention of plagiarizing a source code in an academic environment. A brief overview of the legal standards that apply to computer programs as copyright works is given. Three tools that are most commonly used to detect academic plagiarism - JPlag, MOSS and SIM are described in detail. Esential results of the comparison are shown, the advantages and disadvantages of each of the tools are highlighted, as well as application possibilities in different situations and the proposal for the further development of these tools.

KEY WORDS: Plagiarism, academic plagiarism, source code, plagiarism detection tools, comparative analysis

1. UVOD

Širenje informacija i dostupnost podataka na Internetu značajno su pojednostavili procese učenja i prenošenja vesti kao i novih otkrića. Pored globalizacije znanja i informisanja, sa razvojem Interneta javljaju se nedostaci i opasnosti, naročito na polju zaštite intelektualne svojine. Glavna pretnja autorskim radovima je mogućnost plagiranja njihovih delova ili kompletnog sadržaja. Pod plagijarizmom[1] se podrazumeva nekritičko i namerno preuzimanje sadržaja autorskog rada i predstavljanje kao svog [1]. Zastupljen je u brojnim granama ljudske delatnosti, od obrazovanja i umetnosti, do naučnih i tehničkih oblasti. Plagijarizam je prisutan u najrazličitijim formatima, uključujući tekstove, kompjuterske programe, baze podataka, grafike, crteže i elektronski materijal [2]. Između ostalog predstavlja oblik računarskog kriminala široko rasprostranjenog u savremenom društvu.

Specifični zahtevi studentskih zadataka često uzrokuju zloupotrebe radova kolega kao i kodova dostupnih na Internetu. Plagijarizam izvornog koda jeste najrasprostranjeniji u akademskom okruženju i predstavlja pojavu kopiranja i transformacije programskih zadataka studenata sa minimalnim ulaganjem truda [3]. U ovakvim plagijatima, najčešće postoje sitne izmene u odnosu na originalni kod, kao što su promene naziva varijabli ili neke od metoda, koje značajno otežavaju manuelno prepoznavanje. Pored zloupotrebe zadataka kolega, u akademskoj sredini je zastupljen i oblik plagiranja poznat kao autoplagijarizam. Obuhvata slučajeve u kojima student kopira kompletan kod ili segment koda iz projekta koji je prethodno radio, bez referenciranja i pozivanja na prethodni rad [4].

Povećanje obima plagijarizma, iziskivalo je razvoj alata kojima se ova pojava može prepoznati i sankcionisati u skladu sa odrednicama zakona. Za detekciju plagiranih kodova ne mogu se koristiti alati koji detektuju tekstualni plagijarizam. Za ovo postoje specijalizovani alati koji se razlikuju prema kriterijumu i mehanizmu pretrage. Zasnovani su na radu različitih algoritama i nude različite mogućnosti u detekciji plagijarizma. S obzirom da su danas plagijarizam i borba protiv njega u neprekidnoj trci, iznalaženje novih načina za nekažnjeno plagiranje iziskuje razvoj naprednijih alata za prepoznavanje i zaustavljanje ovakvih malverzacija u savremenom poslovanju, nauci, umetnosti i obrazovanju.

U ovom radu odabrana su tri najčešće korišćena alata za detekciju plagijarizma - JPlag, MOSS i SIM, čije su odlike i performanse detaljno opisane. Definisani su osnovni pojmovi koji predstavljaju predmet ovog rada, dat kratak osvrt na akademsko plagiranje izvornog koda, kao i na autorskopravnu zaštitu računarskih programa. Uvođenje ovih pojmova i razgraničavanje pravnih normi, definicija kompjuterskih programa i kodova iz pravnog aspekta, za cilj ima jasan prikaz situacija koji se mogu smatrati plagijarizmom. Nakon definisanja ovih pojmova, opi-

[1] Dok se pod plagijarizmom (eng. *Plagiarism*) smatra pojava neautorizovanog korišćenja intelektualne svojine, plagiranje podrazumeva same procese koji dovode do nastanka plagijarizma.

sani su pomenuti alati za detekciju plagijarizma i algoritmi na kojima se zasnivaju. Za detaljan opis, iskorišćeni su literaturni podaci i iskustva korisnika u primeni alata.

2. PLAGIRANJE SOFTVERA

Faidi i Robinson su, govoreći o detekciji softverskog plagijarizma na univerzitetima, 1987. godine definisali plagijarizam kao pojavu kopiranja i transformacije programskih zadataka studenata sa minimalnim ulaganjem truda [3]. Koncizna definicija plagijarizma poistovećuje ovaj akt sa nepriznatim kopiranjem dokumenata ili programa, bez navođenja reference [5], takozvano neautorizovano kopiranje dokumenata ili programa [6]. Profesor Aiken sa Stanford univerziteta (*eng. Stanford University*) koji je razvio MOSS, alat za detekciju softverskog plagijarizma, navodi da je plagijat izvornog koda slučaj namernog kopiranja kompletanog koda ili njegovih segmenata, bez referenciranja i navođenja originalnog autora [7].

Pod plagiranjem izvornog koda podrazumeva se ponovna upotreba identičnog koda, autoplagijarizam izvornog koda, kopiranje bez obrade, kopiranje sa minimalnim, umerenim ili ekstremnim obradama, konvertovanje izvornog koda u drugi programski jezik, korišćenje softvera za generisanje kodova (eng. *Code-generator software*), metode za dobijanje izvornog koda napisanog od strane drugog autora i lažno referenciranje [6]. Postoje i takozvane "sive zone" u kojima je problematično definisati plagijarizam. U ovu kategoriju spadaju neki oblici autoplagijarizma i ponovne upotrebe koda u objektno orijentisanim okruženjima [6].

Sve oblike plagijarizma, izuzev prostog kopiranja, karakteriše jedna od dve strategije transformacije. Jedna podrazumeva leksičke, a druga strukturne promene. Pod leksičkim izmenama podrazumevaju se promene formata, imena identifikatora, brojeva linija koda. Za razliku od leksičkog, strukturni plagijarizam podrazumeva poznavanje programskog jezika koje omogućava da se kod raščlani i oprezno izmeni. Ovaj vid plagijarizma je ređi u akademskim krugovima budući da zahteva gotovo isti obim posla kao pisanje programa *de novo* i da studenti podležu plagijarizmu zbog nedovoljnog poznavanja programskog jezika i razumevanja zadatka [6].

Postoje različiti nivoi identičnosti odnosno sličnosti kodova. Sličnost visokog nivoa podrazumeva identične tekstove, a nivo plagijarizma opada sa udaljavanjem od originalnog materijala. Kodovi niskog nivoa sličnosti se u akademskim uslovima, kada studenti rešavaju zadatak vođeni istim uputstima i pravilima, ne smatraju akademskim prestupom. Mere kojima studenti pribegavaju u cilju leksičkog prikrivanja plagijarizma su izmene komentara, formata, identifikatora, promena redosleda operanada, promena tipa podataka, zamena izraza ekvivalentima, dodavanje suvišnih iskaza, promena redosleda međusobno nezavisnih iskaza, promena strukture iteracionih ili selekcionih iskaza, promena kombinovanja originalnih i kopiranih programskih fragmenata [8]. Prema stepenu sličnosti i metodi prikrivanja malverzacije, plagijarizam se može podeliti u osam kategorija:

– I tip plagijarizma podrazumeva istovetnu kopiju originala;
– II tip plagijarizma podrazumeva promene na nivou komentara. Većina detektora plagijarizama ignoriše komentare tako da ovaj tip plagijarizma ne predstavlja ozbiljan problem alatima za detekciju;
– III tip plagijarizma uključuje promenu identifikatora kao što su varijable i nazivi funkcija;
– IV tip plagijarizma je tip u kom se lokalne varijable pretvaraju u globalne i obrnuto;
– V tip plagijarizma označava promenu operanda i matematičkih operacija (na primer $x < y$ prelazi u $y \geq x$);
– VI tip plagijarizma se javlja kada se tipovi varijabli i kontrolne strukture zamenjuju ekvivalentima - iziskuje opreznost zbog mogućeg narušavanja funkcionalnosti koda;
– VII tip plagijarizma podrazumeva izmenu redosleda iskaza. Potreban je poseban oprez da ne dođe do promene funkcije izvornog koda;
– VIII tip plagijarizma javlja se kada se grupe poziva pretvaraju u funkcionalne pozive i obrnuto [9].

Ovom podelom nisu obuhvaćena dva oblika plagijarizma koji se razmatraju zasebno - generisanje izvornog koda korišćenjem softvera za generisanje koda i prevođenje postojećeg programa u drugi programski jezik (*eng. Inter-lingual plagiarism)*[8, 9].

2.1. *Autoplagijarizam i ponovna upotreba koda*

Kao što je pomenuto prilikom definisanja pojma plagijarizma, postoje problematični slučajevi u kojima nije lako napraviti granicu između legalnog i nelegalnog čina ponovne upotrebe koda. U ove „sive zone" plagijarizma zalaze neki slučajevi na ivici plagijarizma i autolagijarizma. Autoplagijarizam (eng. *Self-Plagiarism, Auto-plagiarism*) predstavlja slučaj kada student prekopira kompletan kod ili deo koda iz projekta koji je već radio u novi projekat, bez referenciranja i pozivanja na tu činjenicu. To se odnosi i na kopiranje koda uz minimalne izmene [4, 10, 11].

Mišljenja po ovom pitanju su podeljena. Dok neki profesori smatraju autoplagijarizam delom plagijata, drugi profesori pak smatraju da nema ništa loše u tome da se isti kod ponovno upotrebi na drugim studentskim projektima. Razlog iz kog neki profesori smatraju da autoplagijarizam ne predstavlja slučaj plagiranja koda jeste činjenica da se u objekto-orijentisanom okruženju veliki deo koda ponavlja kroz klase i objekte i čini se neprimernim zabranjivati studentima da ponovo koriste iste segmente koda koji su napisali za potrebe nekog drugog projekta [10]. Slično važi i za ponovnu upotrebu tuđih kodova. Upotreba rada i ideja poznatih u literaturi je fundamentalna u procesu učenja i ovo je oblik upotrebe intelektualne svojine koji se u akademskoj zajednici smatra legitimnim. Znanja i ideje velikih umova temelj su na kom se razvijaju znanja studenata. U ovom slučaju, plagijarizmom se smatra samo upotreba tuđeg koda bez referenciranja i prezentovanje kao svog [7].

U praksi softverskog inženjeringa retko se kreira softver od samog starta odnosno bez upotrebe prethodno generisanih elemenata. Najčešće se prilikom konstrukcije novog softverskog sistema koriste već postojeći artefakti programa (često vezani za validaciju, verifikaciju, dizajn, implementaciju, održavanje,

alate i testiranje). Ovo zahteva veoma pažljiv rad kako bi elementi bili upotrebljeni na kontrolisan način, bez ugrožavanja funkcionalnosti novog softvera [7]. Upotreba već postojećih delova koda značajno povećava produktivnost u generisanju koda, budući da skraćuje vreme potrebno za razvoj nekih elemenata [12]. Programski jezik C++ je tako dizajniran da potpuno podržava ponovnu upotrebu softvera. Rad u ovom programskom jeziku podrazumeva upotrebu biblioteka (poput *iostream*[2]bibilioteke) u kojima se nalaze kodovi koje slobodno koristimo kao deo svog izvornog koda [12]. Razvijena je podrška za ponovnu upotrebu softvera, koja uključuje razvoj dizajna i mogućnosti implementacije softverskih biblioteka, kao i samih alata koji pružaju podršku recikliranju koda. Ove mere se u okviru akademskog plagijarizma mogu zloupotrebiti ili studenti mogu prostom nepažnjom zalutati u neko od nedozvoljenih polja ponovne upotrebe koda [12].

Odgovornost je samog studenata da se obezbedi i ogradi od plagijarizma prilikom direktnog ili indirektnog korišćenja tuđeg rada, odavanjem priznanja originalnoim autoru i ispravnim referenciranjem. Mnogi autori uključeni u akademski rad bavili su se ovom temom i postavili standardne adekvatnog referenciranja koje isključuje plagijarizam [13, 14]. Pored toga, mnogi univerziteti sami propisuju polise i vodiče koji ukazuju na plagijarizam i daju preciznije odrednice ovog fenomena [12]. Ideja stručnjaka koji nastoje da iz ovakvih uputstava isključe nejasnoće jeste pokušaj zajedničkog kreiranja neke forme kodeksa (eng. *Code of Practice*) kojim bi se standardizovale norme koje rasvetljavaju ove granične slučajeve [7]. Cilj kodeksa je pružanje eksplicitnih kategorija koje determinišu plagijarizam i samim tim isključivanje mogućnosti pogrešnih optužbi studenata i olakšavanje objektivne provere, što očekivano rezultuje povećanjem kvaliteta studentskih softvera. Preporučljiva je upotreba odrednica kodeksa i prilikom primene sopstvenih kodova u cilju izbegavanja autoplagijarizma [7].

Kodeks koji preporučuje Paul Gilson 2004. godine [15] ima nekoliko ključnih tačaka:

1) Reciklirani softveri se ne smeju naći u istom fajlu kao softver koji student predaje za ocenjivanje osim u slučaju ovlašćivanja od strane supervizora sa odobrenjem priloženim u dokumentaciji.
2) Svi reciklirani softveri moraju sadržati adekvatno priznanje autoru u priloženoj dokumentaciji i student mora nedvosmisleno ukazati na razliku softvera koje je sam kreirao i softvera koji su reciklirani.
3) Svi reciklirani softveri moraju biti adekvatno testirani.
4) Svi studenti za koje je utvrđeno da su plagirali softver – namerno ili ne, moraju biti kažnjeni [15].

2.2. *Računarski programi kao predmet autorskopravne zaštite*

Kako bi računarski programi bili zaštićeni od malverzacija, neophodno je bilo uvođenje pravnih normi kojima se ta zaštita obezbeđuje. Zaštita intelektualne svojine može se ostvariti kroz autorsko delo, geografsku oznaku porekla, dizajn, patent, mali patent i žig. Različiti vidovi zaštite intelektualne svojine često se jednim imenom nazivaju autorskim i srodnim pravima. Ovi oblici zaštite obezbeđuju stvaraocima originalnog dela prepoznavanje i finansijski benefit za predmet njihovog rada [16]. Za cilj imaju zaštitu dela od kopiranja, obrade, objavljivanja, emitovanja i drugih vidova korišćenja bez ovlašćenja od strane autora. Isključivo autor i ovlašćena lica imaju pravo na upotrebu zaštićenog dela. Jedan od izazova prilikom definisanja zakona intelektualne svojine jeste potreba za opštim definicijama u okviru zakona koje ostaju primenljive sa razvojem tehnologije [17, 18].

Autorsko delo je intelektualna tvorevina koja nastaje kao rezultat duhovnog stvaralaštva tvorca [19]. Da bi neko delo bilo zaštićeno kao autorsko, potrebno je da bude izraženo u nekoj formi, da bude originalno, odnosno da u tom obliku prethodno nije postojalo, da je rezultat individualnog rada autora ili navedene grupe autora/organizacije/institucije, da bude predstavljeno javnosti, da su ispunjene formalnosti – oznaka *copyright* ©, depozit, registracija, klauzula o proizvodnji [19]. Od ovih uslova se sve češće u raznim autorskopravnim sistemima odstupa pa tako nije nužno da forma autorskog dela bude fizički materijalizovana, u nekim zemljama (između ostalih i našoj) objavljivanje dela nije nužan preduslov za njegovu zaštitu, a uslov ispunjavanja formalnosti se sve češće napušta.

Procenjuje se da je u SAD samo 25% softvera dostupnih na tržištu donosi koristi autorima i vlasnicima. Ovaj stepen prestupa donosi SAD godišnje gubitke u stotinama miliona dolara. Problemi ovakve prirode izazivaju seriju drugih problema na polju međunarodnih trgovinskih i poslovnih odnosa. Upravo ovo je navelo stručnjake iz oblasti informatike i prava da sinergističkim radom uvedu računarske programe u sistem autorskopravne zaštite. Računarski programi su definisani kao autorsko delo, budući rezultat stvaralačkog napora jednog ili više autora i zaštićeni na isti način kao književna dela [19, 20]. U pojedinim nacionalnim zakonodavstvima, programi se tretiraju kao prevodi, naučno-tehnički prikazi ili kao posebna autorskopravna dela [19]. Sama ideja računarskog programa nije pravno zaštićena, a forma u kojoj se štite računarski programi podrazumeva set instrukcija koje se ubacuju u hardvere [19].

U svakodnevnoj praksi pojmovi softver i računarski program su nekad izjednačeni, ali ovi pojmovi se suštinski i pravno razlikuju. Softver je opštiji pojam koji podrazumeva nematerijalni sistem kakav je računarski program, ali i niz datoteka i prateće dokumentacije kao što su priručnici i uputstva za upotrebu, razumevanje i opsluživanje programa (eng. *Manual*). Kada je reč o računarskom programu, postoji definicija u užem smislu prema kojoj program obuhvata niz instrukcija kojima se upravlja obradom podataka, dok u širem smislu, pored niza instrukcija, u računarski program se ubrajaju i programski koncept - programska zamisao rešavanja određenog problema i algoritmi koji predstavljaju seriju instrukcija ili proceduralnih koraka u rešavanju problema [19].

Zaštita autorskih prava odnosi se na računarski program dok se propratna dokumentacija može zasebno zaštititi u okviru neke od formi autorskog dela. Najčešće se tretira kao uput-

[2] Input/Output biblioteka koja sadrži *izlazni tok* i funkcije koje omogućavaju ispis podataka na ekran.

stvo i štiti se samo ukoliko je proizvođačka i zadovoljava uslove propisane zakonom. Poseban predmet zaštite je i korisnički interfejs koji se sastoji iz ikona [19, 21, 22].

Pored pomenute razlike između softvera i računarskih programa, trebalo bi razlikovati programe, programske jezike i oblike u kojima se program nalazi. Računarski programi mogu se pojaviti u izvornoj, objektnoj i izvršnoj verziji, odnosno mogu se naći u nekoliko oblika: izvornom, objeknom (prevedenom), izvršnom i pseudo kodu. Za pravnike je naročito važna razlika između izvornog i objektnog koda odnosno između izvorne i objektne verzije [19].

Programski jezik je definisani set simbola za upravljanje računarom preko definisanih pravila. Budući da je svaki program napisan u nekom programskom jeziku, isti se smatra delom računarskog programa [19]. Česti su plagijati prevođenja koji podrazumevaju da se kod prevodi iz jednog u drugi programski jezik, dok je ideja potpuno preuzeta. Ovakve situacije su veoma značajne za pravo pa je značajna i precizna definicija programskog jezika [19]. **Izvorni kod** (eng. *Source code*) ili **izvorna verzija programa** (eng. *Source version*) je program u formi u kojoj ga piše tvorac [19]. Prevođenjem programa u mašinski odnosno binarni jezik nastaje **objektni**, odnosno prevedeni kod (eng. *Object code / Machine code*) ili **objektna verzija programa** (eng. *Object code version*). Ovo je jedina forma programa koja trpi izmene i prepravke [19]. **Izvršni kod** odnosno **verzija** (eng. *Task version* ili *Execute version*) je jedina verzija programa koja se zaista izvršava. Ima praktični značaj s obzirom da su druge verzije pripremne, ali nije značajna za autorskopravnu zaštitu [19]. Još jedna poznata forma je **pseudo kod** (eng. *Pseudocode*) u kom su instrukcije pisane simbolima i koji pre puštanja u rad zahteva prevođenje na mašinski kod korišćenjem kompajlera [19, 23].

S obzirom na razlike između oblika odnosno verzija računarskih programa, ističe se značaj izvornog koda u pravnoj zaštiti. Objektni kod ne pruža mogućnost unošenja značajnijih izmena u računarski program. Za ovakve izmene, neophodan je pristup izvornom kodu. Lak pristup izvornom kodu pored mogućnosti ispravljanja bagova i usavršavanja programa, pruža mogućnost zloupotrebe u vidu piraterije ili plagijata. Iz tog razloga, izvorni kod je najosetljivija verzija računarskog programa kada je autorsko pravo u pitanju [19].

3. ALATI ZA DETEKCIJU PLAGIJARIZMA

Prvi poznati sistem za detekciju plagijarizma imao je algoritamski pristup. Razvio ga je Otenštejn 1976. godine [24]. Koristio je Halstedovu metriku za poređenje dva programa u programskom jeziku Fortran. U algoritmima koji rade na principu Halstedove metrike poredi se broj jedinstvenih operatora, broj jedinstvenih operanada, ukupni broj operatora i operanada. Ukoliko se sva četiri podatka poklapaju u dva programa, jedan od njih je plagijat [25].

Alati za detekciju plagijata izvornog koda mogu se grubo podeliti u dve kategorije – alati koji se baziraju na odlikama (eng. *Feature-based*) i na strukturi koda (eng. *Structure-based*) [25]. Uglavnom se alati novijeg datuma baziraju na strukturi koda, a u prethodnih nekoliko godina razvijaju se i takozvani hibridni alati koji mešovitim pristupom omogućavaju detekciju plagijata [25]. Prva kategorija alata podrazumeva prevođenje fajla izvornog koda u niz brojeva i poređenje tako dobijenih sekvenci. Zasnivanje alata za detekciju plagijarizma na odlikama, specifičnim atributima programskog jezika, nije dovoljno precizan osnov za poređenje kodova [26]. Iz tog razloga alati koji su danas u najširoj upotrebi zasnivaju se na poređenju strukture izvornog koda programa. Među ovim alatima postoje neki dostupni na internetu besplatno kao što su JPlag, MOSS, CodeSuite, kao i lokalni alati dostupni u Desktop varijanti - YAP3, Plaggie, Sherlock, SIM, Marble, CPD [9].

Sami algoritmi za detekciju plagijarizma izvornog koda se mogu klasifikovati prema mehanizmu pretrage na nekoliko grupa. Najpregledniju klasifikaciju dali su Roj i Kordi [28] 2007. godine. U knjizi u kojoj se bave alatima za detekciju plagijarizma, razlikuju algoritme koji funkcionišu na osnovu:

- Teksta odnosno niza (eng. *Text/String-based*) – program se posmatra kao skup nizova i traži se tekstualno podudaranje segmenata koda. Pretraga je brza, ali ograničena je na isključivo leksičku, a ne i strukturnu podudarnost.
- Tokena (eng. *Token-based*) – ceo kod se prevodi u sekvencu tokena. Ova sekvenca ne sadrži komentare i identifikatore pa su ovakvi alati osetljiviji na jednostavne izmene u tekstu. Ovi sistemi su izuzetno su uspešni u detekciji akademskih plagijata.
- Drveta izvođenja (eng. *Tree-based*) – program se prikazuje u vidu apstraktnog sintaksnog drveta. Poređenjem dva ovako prikazana programa mogu se detektovati sličnosti veoma niskog nivoa. Potpuno eliminiše nazive varijabli i identifikatore tako da u osnovi drveta ostaje celokupna informacija o kodu.
- Grafika (eng. *Program Dependency Graph (PDG)-based*) – uzima u obzir semantičku informaciju koda. Snažni su u prepoznavanju preraspoređenih iskaza**,** insercija i delecija delova koda, kao i isprepletanih kodova, ali ne za velike programe.
- Metrike (eng. *Metrics-based*) – dodeljuju segmentima koda određene vrednosti (npr. broj petlji, uslovnih izraza ili promenljivih) i onda njih međusobno porede umesto poređenja samog koda. Detekcija je brza, ali uz opasnost od lažno pozitivnih vrednosti.
- Hibridnog pristupa (eng. *Hybrid approaches*) – kombinuje neke od prethodno pomenutih pristupa u cilju eliminisanja nedostataka svakog od njih [28, 29].

Pored mehanizma detekcije plagijarizma, brojne su karakteristike koje se mogu koristiti za poređenje ovih alata u cilju ispravnog odabira za upotrebu u realnom problemu. Karakteristike koje se odnose na kvalitativna svojstva alata kao što su – primenljivost u određenom programskom jeziku ili dostupnost u okviru Web ili lokalnih servisa, početni su parametar za izbor alata. Ovo je poređenje svojstava alata (eng. *Feature comparison*). Međutim, ukoliko nekoliko alata može u datim uslovima biti upotrebljeno za rešenje nekog problema, porede se kvantitativne osobine, koje se odnose ne na sam mehanizam rada, već na rezultate pretrege. U pitanju je poređenje perfor-

mansi (eng. *Performance comparison*) budući da se alati porede prema svom učinku [26].

Mnogi revijalni radovi [6, 26, 29,30] bave se poređenjem alata za detekciju plagijarizma izvornog koda i sistematično prikazuju razlike između njih. Neke od osnovnih odlika najčešće primenjivanih alata za detekciju plagijarizma izvornog koda prikazane su u Tabeli 1.

Tabela 1. ***Osnovne karakteristike alata dostupnih u literaturi za detekciju plagijarizma izvornog koda [2, 9, 26, 31].***

Osobina	JPlag	Marble	MOSS	Plaggie	SIM
Tehnika/ Algoritam	Greedy String tiling/ Token	Strukturni analizator	Winowing algoritam	Greedy String tiling/ Tokens	Leksički analizator
Interfejs	Grafički prikaz	GUI	Grafički prikaz	Grafički prikaz	GUI
Osnovan	1996	2002	1994	2002	1989
Osnivač	Gvido Malpohl	Jurrian Hage	Aiken i sar.	Ahtiane i sar.	Dik Grun

Nijedan od ovih alata ne može direktno dokazati prisustvo plagijarizma. Informacija koju daju je mera sličnosti između programa. Analizu rezultata pretrage vrši čovek koji je u stanju da proceni da li je sličnost dva koda posledica upotrebe standardnih naučenih metoda za rešavanje određenih zadataka, grupnog rada studenata, slučajnosti ili pak plagijarizma [26].

3.1. *JPlag*

JPlag je razvio Guido Malpohl na Institutu za tehnologiju u Karlsrue (Nemačka) 1996. godine. Počeo je kao studentski projekat da bi nekoliko meseci kasnije evoluirao u on-line sistem. JPlag je 2005. pretvoren u Web service, što znači da nije vršena instalacija na lokalni računar, nego su se korisnici putem mejla prijavljivali za besplatan JPlag nalog, a onda sa zvaničnog sajta preuzimali *Java Web Start Client* preko kog su se logovali. Alat je na ovaj način funkcionisao sve do 2016. godine kada je prestala podrška za korišćenje JPlag-a on-line [32]. Novi korisnik može preuzeti JPlag sa Github[3]-a. Više ne postoji mogućnost registracije i korišćenja preko Web servisa. Moguće je i korišćenje JPlag-a kao *Moodle plugin*-a. Zahteva minimum verziju 7 Jave [33].

Program je dizajniran tako da pronalazi sličnosti između višestrukih setova fajlova (klasa) izvornog koda. Ne upoređuje nizove tokena po principu *bit by bit*, već je u fokusu sintaksa programskog jezika kao i programska struktura. Algoritam koji JPlag koristi za poređenje dva niza tokena je "pohlepni" algoritam (eng. *Greedy String Tiling)*, koji se u programiranju najčešće koristi za detekciju dupliranih stringova. Algoritam je dizajniran tako da kod poređenja dva niza tokena, uvek pokušava pronaći skup podnizova koji su jednaki. JPlag konvertuje kompletan kod u nizove tokena koji se nazivaju "*token strings*". Ovakvi nizovi tokena predstavljaju strukturu programa i samim tim se može reći da JPlag spada u kategoriju alata koji se baziraju na sekvenci tokena (eng. *Token-based*). Svaki token iz jednog izvornog koda programa se može upariti sa najviše jednim tokenom iz drugog izvornog koda programa. Zbog ovakvog načina funkcionisanja algoritma, nije moguće upariti delove izvornog koda koju su duplirani u drugom izvornom kodu, odnosno plagijatu. Podnizovi se traže bez obzira na položaj u klasama, pa se na taj način eliminiše eventualna promena rasporeda u plagiranom izvornom kodu. Duži nizovi koji se porede, smatraju se boljim od kratkih koji predstavljaju sumnju, jer je moguće da se radi o slučajnim sličnostima kao što su slova i delovi reči [32, 34].

Kao posledica ovog mehanizma poređenja, JPlag je snažan u prepoznavanju pokušaja prikrivanja sličnosti između plagiranih fajlova. Podržava otkrivanje plagijata u Java, C#, C, C++ programskim jezicima, kao i rad sa Scheme i prirodnim jezikom [35, 36]. Treba napomenuti da je sistem *Ephours*[4] zasnovan na konceptu JPlag alata. JPlag je brz, pouzdan i rezultati dobijeni korišćenjem ovog alata ukazuju na pouzdano pronalaženje plagijata sa minimalnim odstupanjem. Vreme izvršenja programa je samo nekoliko sekundi za obim od 100 klasa sa po više stotina linija koda unutar svake. Ovaj alat je igrao veliku ulogu u nekoliko slučajeva zaštite intelektualne svojine u kojima su ga forenzički eksperti uspešno koristili za pronalaženje dokaza [33].

Često dolazi do pogrešnog tumačenja da JPlag poredi kod sa sadržajem na Internetu. Alat je prvenstveno dizajniran da pronađe sličnosti između studentskih zadataka. Budući da je najčešći oblik plagijarizma u studentskim radovima vezan za razmenu rešenja između samih studenata, ovakav vid provere je najpotrebniji [33].

U nastavku je dat primer pozivanja programa za poređenje studentskih programa napisanih u Java 1.7. verziji. Prilikom poređenja studentskih programa, svaki program treba da bude smešten u poseban folder (ime i prezime i broj indeksa studenta). Svi pojedinačni zadaci se nalaze u jednom glavnom folderu koji nosi ime, npr ''Vežba 1''.

Za pokretanje JPlag alata, dovoljno je unošenje sledećih komandi:

java -jar jplag-yourVersion.jar -l java17 -r /tmp/jplag_results_exerise1/ -s /path/to/exercise1

- **-l-** java17 upozorava JPlag da koristi Java programski jezik, minimalna verzija 1.7
- **-s-** obaveštava JPlag da funkcioniše u okviru podfoldera; kada preuzmemo Java projekat, naići ćemo na podfoldere poput - student1/src/
- **-r-** /tmp/jplag_results_exercise1 obaveštava JPlag da čuva rezultate u direktorijumu - /tmp/jplag_results_exercise1

Neophodno je precizirati u kom programskom jeziku je napisan dati zadatak budući da pokretanje JPlag alata bez nazna-

[3] GitHub je hosting servis. Predstavlja mesto gde programeri mogu da dele svoj izvorni kod i priče sa ostatkom sveta.

[4] http://www.ephorus.com/ - Program koji otkriva da li su radovi studenata plagijati ili su autorska dela

ke u komandnoj liniji podrazumeva pokretanje svih dostupnih jezika i drugih opcija. Na primer, ukoliko je potrebno proveriti procenat podudaranja koda u C++ programskom jeziku, neophodno je da se precizira *-l c/c++* kao opcija programskog jezika.

U sledećem primeru se može zaključiti da Java izvorni kod ima 8 tokena, i to početak klase (*beginclass*), zatim početak metode (*beginmethod*), inicijalizaciju promenljive *count* koja je identifikator (*assign*), zatim početak *While* petlje (*beginwhile*) koja predstavlja ključnu reč (eng. *Key word*), pa onda kraj *While* petlje (*endwhile*), ispisivanje vrednosti *count* promenljive (*apply*), nakon toga kraj metode *Main* (*endmethod*)i kraj klase *Count* (*endclass*).

```
1 public class Count {                          BEGINCLASS
2   public static void main(String[] args)      VARDEF,BEGINMETHOD
3       throws java.io.IOException {            (NEMA TOKENA)
4       int count = 0;                          VARDEF,ASSIGN
5       while (System.in.read() != -1)          APPLY,BEGINWHILE
6           count++;                            ASSIGN,ENDWHILE
7       System.out.println(count+" chars.");    APPLY
8       }                                       ENDMETHOD
9 }                                             ENDCLASS [32]
```

Kao u primeru iznad, sa klasom *Count*, na trećoj liniji koda, neki tokeni se takođe ne smatraju korisnima. Tako recimo osnovni skup tokena za Javu u potpunosti ignoriše *Exception*-e. Broj linije se smešta u sam token čime je omogućeno vraćanje unazad i određivanje mesta u izvornom kodu programa, odakle je potekla određena sličnost.

JPlag podržava istragu detekcije plagijata svojim jedinstvenim korisničkim interfejsom. Kao što je već pomenuto, JPlag nudi jedinstven, efikasan i veoma intuitivan korisnički interfejs. Rezultati se prikazuju u obliku histograma na HTML stranicama koje JPlag automatski generiše, prilikom izvršavanja. Ovakve stranice sadrže listu fajlova koji se podudaraju. U gornjem levom uglu generisane HTML stranice je procentualno izražena sličnost između programa. Konačni rezultati su prikazani u dva odvojena prozora, pa se klikom na hiperlinkove iz histograma poravnava leva sa desnom stranom ekrana na mestu gde je kod isti, tj. gde je kod plagiran. JPlag nudi *side-by-side* pregled fajlova koji se analiziraju, gde su segmenti koji se podudaraju iz oba fajla označeni istom bojom [31, 32].

3.2. *MOSS*

Naziv ovog alata predstavlja akronim za merenje sličnosti softvera (*eng. Measure Of Software Similarity*). Razvila ga je grupa profesora Aleksa Aikena[5] sa Berkli univerziteta 1994. godine [7]. HTML interfejs alata MOSS produkovan je od strane autora alata Jplag, Gvido Malpohl [9]. Može se koristiti isključivo za detekciju plagijarizma izvornog koda, dok preklapanja u tekstualnim dokumentima ne može pronaći [12].

Za potrebe ovog alata, profesor Aiken sa saradnicima, razvio je algoritam za takozvani "*document fingerprinting*" što doslovno predstavlja "*otisak prsta dokumenta*". Algoritam je poznat pod nazivom Winowing i detektuje strukturno preklapanje kodova. Stringovi na koje se kod raščlanjuje u svim alatima baziranim na nizovima, dodatno se razdvaja na granične substringove (*k-gram*) čiju dužinu (*k*) zadaje korisnik i koji se izvode iz osnovne strukture koda. Isecanjem *k-gram* podstringova dobija se skup sekvenci među kojima se biraju "otisci prsta" izvornog koda. Odabir ovih isečaka vrši se tako da je mogućnost kolizije veoma mala, što omogućava da preklapanje samo jednog ili nekoliko ovakvih isečaka garantuje preklapanje celog *k-gram* podstringa, odnosno predstavlja jakog kandidata za plagijat [2, 13, 30].

Upravo zbog svog mehanizma pretrage, Winowing algoritam, odnosno MOSS alat, karakteriše izuzetno nizak stepen greške u vidu lažno negativnih rezultata. Garancija za detekciju preklapanja nije apsolutna upravo zbog odabira isečaka podstringa koji predstavljaju glavni parametar pretrage, ali pažljivim odabirom od strane stručnjaka, ovaj nedostatak alata može se svesti na minimum [13].

MOSS se može koristiti za analizu kodova napisanih u brojnim programskim jezicima: C, C++, Java, C#, Python, Visual Basic, JavaScript, FORTRAN, ML, Haskell, Lisp, Scheme, Pascal, Modula2, Ada, Perl, TCL, Matlab, VHDL, Verilog, Spice, MIPS assembly, a8086 assembly, HCL2 [26]. Ima mogućnost proširivanja i adaptacije na druge programske jezike, isključivo od strane autora. Alat je otvorenog koda. Dostupan je u vidu Web servisa (https://theory.stanford.edu/~aiken/moss/), a za pristup i korišćenje ovog alata neophodno je kreiranje naloga uz odobrenje koje se može dobiti kontaktiranjem autora (e-mail: moss@moss.stanford.edu). Trenutna validna skripta koju MOSS nudi na svom sajtu je u verziji 2.0. Nakon registracije, MOSS novom korisniku šalje poslednju verziju skripte kao i jedinstveni ID. U bilo kom trenutku je moguće preuzeti skriptu sa zvaničnog sajta, samo je potrebno korigovati na mestu gde je definisan korisnički ID. Proces izvršavanja je jednostavan. Skriptu je potrebno prekopirati u novi tekstualni fajl i snimiti sa ekstenzijom **.pl** (konkretno *moss.pl*). Da bi bilo moguće izvršavanje skripte potrebno je instalirati *Perl* koji može da se preuzme sa adrese - http://strawberryperl.com/.

Jednostavan primer izvršavanja MOSS alata bi bio: *perl moss.pl -l csharp File1.cs, File2.cs, File3.cs...* gde se pored standardnog izvršavanja skripte poziva i komanda ***–l*** kojom se definiše u kom programskom jeziku su pisani fajlovi koji se porede. U ovom slučaju to je C# programski jezik. Nakon toga navode se imena fajlova zajedno sa ekstenzijama koji se analiziraju. Prvo se proverava struktura fajlova i ukoliko je sve u redu, fajlovi se šalju na MOSS-ov server. Nakon *upload*-a fajlova, sledi odgovor servera u vidu URL adrese. Adresa je u formatu: http://moss.stanford.edu/results/XXXXXXXXX. Poslednjih devet cifara su jedinstvene i svaki put kada se novi fajlovi okače na server ti brojevi se menjaju [7].

MOSS ima mogućnost isključivanja templatnih kodova, što znači da korisnik može zadati tip podatka (eng. *Base file*) koji se potpuno ignoriše prilikom pretrage preklapanja. Ovo

[5] Od 2003. godine, profesor Aiken predaje na Stanford univerzitetu.

može biti od koristi profesorima, koji svojim studentima daju instruktorske kodove za koje je velika verovatnoća da će ih veći broj studenata iskoristi, a što se ne može smatrati prekršajem. Ovim se uspešno eliminiše jedan od izvora lažno pozitivnih rezultata pretrage koji potiče od legitimne razmene kodova. Takođe ima mogućnost isključivanja malih podataka za koje je velika verovatnoća nenamernog pojavljivanja u nezavisnim kodovima [26]. Ovaj program omogućava korisniku da u jednom danu ispita 75 do 120 programa, svaki dužine nekoliko hiljada linija koda [15].

Kao i većina drugih alata, MOSS ne omogućava potpuno automatsko detektovanje plagijarizma. Na korisniku ostaje da obeležene i alarmirane delove sekvence koda proceni kao plagijat ili ne, u zavisnosti od očekivanja, prethodnog znanja, dozvoljene literature i smernica datih studentu prilikom pisanja koda. Pored neophodne uloge čoveka u analizi rezultata pretrage, uloga alata je nezamenljiva budući da umnogome štedi vreme profesora koji nastoje da ispitaju eventualne plagijate [26].

Rezultati pretrage se prikazuju u vidu HTML prezentacije sa linkovima i integrisanim HTML editorom koji omogućava navigaciju kroz rezultate [26, 27]. Za svaki par programa, kratak pregled rezultata sadrži broj tokena koji se poklapaju, broj linija koda koje se poklapaju i ukupan procenat izvornog koda koji se preklapa [15]. Pored ovog pregleda, izveštaj sadrži i detaljniju analizu plagijarizma sa obeleženim regionima preklapanja [9]. Po završetku analize, na Web serveru dostupan je link do Web strane sa rezultatima kojoj ne može pristupiti drugi korisnik. Već nakon 14 dana, stranicama sa rezultatima više se ne može pristupiti. Ovim merama, mogućnost zloupotrebe svedena je na minimum [26, 27].

3.3. SIM

SIM (eng. *Software Similarity Tester*) je efikasan alat za detekciju plagijarizma koji funkcioniše i izvršava sve naredbe putem komandne linije. Ovaj alat nema svoj grafički korisnički interfejs (eng. *GUI*) i zbog toga spada u grupu alata koji koriste korisnički interfejs komandne linije (eng. *Command line interface*). Trenutna verzija je 2.89. Napisao ga je Dick Grune[6] u *C* programskom jeziku 1989. godine na *Slobodnom uziverzitetetu u Amsterdamu* (*Vrije University Amsterdam*) na kome je radio kao profesor. Važno je napomenuti da ovaj alat već odavno nije aktivno podržan, ali izvorni kod je još uvek dostupan javnosti [9, 31, 37].

Detektuje plagijarizam u projektima koji su pisani u C, Java, Pascal, Modula-2, Lisp i Miranda programskim jezicima [37]. Inicijalno SIM nije nastao kao alat za detekciju plagijata, nego kao alat za pronalaženje istih delova izvornog koda unutar velikih programa kao što su operativni sistemi, kompajleri i sl. radi smanjivanja veličine izvornog koda i optimizacije programa [37].

SIM čita fajlove i traži segmente koda koji su slični; dva segmenta programa se smatraju sličnim ako se razlikuju samo po nekim karakteristikama kao što su raspodela koda, komentarima, identifikatorima, sadržaju brojeva, stringova. Ukoliko alat pronađe sličnosti, izdaje izveštaj o detektovanim sličnostima u okviru standardnog izlaza sa brojem značajnih tokena datim između uglastih zagrada. SIM konvertuje izvorni kod pomenutih 6 programskih jezika u nizove tokena, ali na drugačiji način od JPlag programa. Koristi prilagođen algoritam da bi pronašao najduži zajednički podniz (eng. *Longest common sub-string*) koji nije osetljiv na raspored linija koda u programu, za razliku od metode najduže zajedničke podsekvence (eng. *Longest common subsequence*). Algoritam je baziran na šablonu podudaranja uzoraka (eng. *Pattern-matching*) koji se zasniva na radu projekta ljudskog genoma (eng. *Human Genome Project*) [9, 38, 39].

Može se koristiti za pronalaženje kopiranih delova koda u navodno nepovezanim programima (sa komandama -s ili -S) ili za pronalaženje slučajno dupliranih kodova u većim programima (komande -f ili -F). U slučaju kada je separator / prisutan na listi ulaznih fajlova, fajlovi su podeljeni u dve grupe - u gupu novih i starih fajlova. Ukoliko ne postoji / ili | svi fajlovi se smatraju novim. Stari fajlovi se nikada ne porede jedni sa drugima [37].

Spisak svih komandi koje se mogu pozvati, kao i pravilan raspored prilikom poziva izgleda ovako:

sim_c [- [defFiMnpPRsStv] -r *N* -t *N* -w *N* -o F] fajl ... [[/]] fajl...]

SIM se poziva jednostavnom komandom *sim_c.exe *.c* sa komandne linije. Ova komanda predstavlja poziv na analizu svih fajlova iz foldera koji se analiziraju. Ističe duplikate koda u direktorijumu. Poziv *sim_c.exe -f -F *.c* može dalje tačno odrediti duplikate koda. ***-p*** komanda daje rezultate u sledećem obliku: *Fajl1 sadrži x% od Fajla2*, gde x% teksta Fajla1 može biti pronađeno u Fajlu2. Treba voditi računa da u ovom rezultatu ne važi jednakost; Moguće je da jedan od fajlova sadrži 100% teksta drugog fajla, dok drugi fajl sadrži samo 1% teksta prvog ukoliko se njihove dužine dovoljno razlikuju. Komanda (***-P***) prikazuje glavne kontributore za svaki fajl pojedinačno. Na ovaj način se pojednostavljuje identifikacija setova fajlova A[1]...A[n], gde je jasno da je u pitanju niz fajlova. Prag može biti definisan korišćenjem opcije ***-t***. Nivo detalja u prepoznavanju teksta (tzv. granularnosti) je definisana opcijom ***-r***. Komanda ***-r***kontroliše broj jedinica koje sačinjavaju jedno pokretanje programa. Za programe koji porede kod programskog jezika, jedinica je leksički token u tom programskom jeziku. Komentari i standardni materijal u klasama se ignorišu i svi karakterni nizovi se smatraju jednakim. Komande ***-s*** i ***-S*** kontrolišu koji će se fajlovi porediti. Ulazni fajlovi su podeljeni u dve grupe i to - stari i novi. U nedostatku ovih kontrolnih opcija, program poredi 4 nova sa 6 starih fajlova [37].

Na sledećem primeru dat je broj tokena u jednoj *for* petlji:

for (int i = 0; i < max; i++); će biti zamenjena nizom tokena: *TKN_FOR TKN_LPAREN TKN TKN_ID_I TKN_EQUALS TKN_ZERO* i tako dalje [38]. Broj tokena za ključne reči i specijalne simbole je već unapred predefinisan. Elementi kao što su zarezi, operatori, identifikatori i sl. SIM prihvata kao nepromenjene tokene. Svi elementi koji ne utiču na rad samog pro-

[6] Više o autoru na: http://www.dickgrune.com/

grama poput komentara, prelaska u novi red, praznog prostora između linija koda i sl. se zanemaruju. Ovakav pristup zahteva da modul koji se bavi konverzijom izvornog koda u tokene u sebi sadrži leksički analizator koji je vezan za programski jezik u kom je izvorni kod koji se poredi napisan. Hipotetički, izvorni kod veličine 4.000 karaktera, aproksimativno će biti sveden na otprilike 200 tokena. Nakon konverzije u tokene počinje poređenje zajedničkih delova [40].

Ukoliko je procenat podudaranja manji od 20%, SIM smatra da je u pitanju koincidencija i takav rezultat može da se zanemari, dok se rezultat sličnosti od 30% smatra značajnim. Pretpostavka je da su rezultati između 20 i 30% problematični, sumnjivi rezultati i da je verovatno deo njihovog sadržaja plagiran [37].

4. ZAKLJUČAK

U informacionim tehnologijama, plagiranju je posebno podložan izvorni kod. JPlag, MOSS i SIM su samo neki od brojnih alata koji se mogu uspešno primeniti u detekciji plagijarizma, a s obzirom na eksponencijalan rast količine informacija i razvoj alternativnih i latentnijih oblika plagiranja, alate za njihovu detekciju očekuje usavršavanje i ekspanzija.

Za odabir najpogodnijeg alata za detekciju plagijarizma određene vrste, neophodno je poređenje njihovih kvalitativnih i kvantitativnih karakteristika. Nakon detaljnog poređenja alata, može se izvesti zaključak da nisu jednako funkcionalni u detekciji plagijarizma. Sumiranjem prikazanih podataka o pomenutim alatima, može se zaključiti da su JPlag i MOSS daleko pogodniji od alata SIM za detekciju akademskog plagijarizma izvornog koda.

Upravo zbog jednostavnosti upotrebe i jasnog i detaljnog prikazivanja rezultata, JPlag je najčešće upotrebljavan alat. MOSS, koji se u ovim parametrima oslanja na rešenja alata JPlag, takođe pruža adekvatno prikazane rezultate i jednostavnost u radu. Glavna razlika, tip algoritma na osnovu kog pretražuju podudarnosti, jeste osnov prilikom odabira jednog od ova dva alata. Presudne su same potrebe korisnika, a poželjna je i upotreba dva ili više alata i poređenje dobijenih rezultata. Za razliku ova dva alata, SIM nije dovoljno jednostavan za korišćenje i prikaz rezultata zahteva detaljniju analizu. Pored toga, nema mogućnost isključivanja templatnog koda i malih fajlova koji se često nalaze u studentskim radovima kao posledica instrukcija samih profesora koje se mogu legitimno primeniti. Usled ovih ograničenja, alat nije pogodan za korišćenje u akademskoj sredini u kojoj je često neophodno poređenje velikog broja radova i u kojoj je poželjno da ovaj proces ne oduzima mnogo vremena profesorima. Može biti značajan stručnjacima u slučaju poređenja malog broja fajlova jer daje iscrpne informacije i svako prisutno preklapanje, bez mogućnosti isključenja bilo kojih segmenata. S obzirom na mogućnost proširenja za različite programske jezike, stručnjacima može biti značajan u proveri prisustva autoplagijarizma pre objavljivanja i distribuiranja novih programa, odnosno objavljivanja radova.

Svaki od opisanih alata ima manje ili veće nedostatke koji se mogu prevazići u predstojećim verzijama. Za dalje usavršavanje alata JPlag, od posebnog značaja je uvođenje mogućnosti jezičkih proširenja, kod alata MOSS to je povećavanje stepena bezbednosti fajlova kriptovanjem sadržaja koji se šalje na Web browser, dok je za alat SIM najznačajnije uvođenje grafičkog prikaza rezultata.

LITERATURA

[1] EC-Council. Computer Forensics: Investigating Network Intrusions and Cybercrime (CHFI). 2011. EC-Council Press, Volume 4.

[2] Gondaliya T, Joshi H, Joshi H. Source Code Plagiarism Detection 'SCPDet': A Review. International Journal of Computer Applications. 2014; 105 (17): 27-31.

[3] Faidhi JAW, Robinson SK. An empirical approach for detecting program similarity and plagiarism within a university programming environment. Computers & Education. 1987; 11 (1): 11-19.

[4] Cosma G. An Approach to Source-Code Plagiarism Detection and Investigation Using Latent Semantic Analysis. PhD Thesis. Warwick: University of Warwick, Department of Computer Science, 2008.

[5] Lillard TV, Garrison CP, Schiller CA, Steele J. Digital forensics for network, internet, and cloud computing. Burlington: Syngress Publishing, 2010.

[6] Joy M, Luck M. Plagiarism in programming assignments. IEEE Transactions of Education. 1999; 42 (2): 129-133.

[7] Dostupno na: https://theory.stanford.edu/~aiken/moss/; Pristupano: 28.6.2016.

[8] Arwin C, Tahaghoghi CMM. Plagiarism detection across programming languages. Proceedings of the 29th Australasian Computer Science Conference (ACSC 2006), Hobart, Australia. CRPIT. 2006; 48 (277-286).

[9] Martins V, Fonte D, Henriques PR, da Cruz D. Plagiarism detection: A tool survey and comparison. OpenAccess Series in Informatics. 2014; 38: 143–158.

[10] Cosma G, Joy M. Source-code plagiarism: a UK Academic Perspective. 7th Annual Conference of the HEA Network for Information and Computer Sciences, Dublin. 2006; 116-120.

[11] Ćapara D, Grubor G, Regodić D, Ristić N. Plagiarism and copyright protection on the internet. 14. Međunarodni naučni skup Sinergija, 2013.

[12] Juričić V, Jurić T, Tkalec M. Performance evaluation of plagiarism detection method based on the intermediate language. INFuture2011: "Information Sciences and e-Society", Zagreb, 2011.

[13] Schleimer S, Wilkerson DS, Aiken A. Winnowing: Local Algorithms for Document Fingerprinting. Conference: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, 2003.

[14] Bowyer KW, O' Hall L. Experience using "MOSS" to detect cheating on programming assignments. Frontiers in Education Conference 1999; 3: 18-22.

[15] Bowyer KW, O' Hall L. Reducing effects of plagiarism in programming classes. Journal of Information Systems Education. 2001; 12 (3): 141-148.

[16] Zvanična prezentacija WIPO. Dostupno na: http://www.wipo.int/portal/en/index.html; Pristupano: 2.5.2016.

[17] Bainbridge D, Computers and the Law. London: Pittman Publishing, 1990.

[18] Andelin J, Curlin JW, Winston JD, Hironaka AM. Computer Software and intellectual Property - Background Paper, U.S. Congress, Office of Technology Assessment, OTA-BP-CIT-61. Washington, DC: U.S. Government Printing Office, 1990.

[19] Drakulić M. Osnovi kompjuterskog prava. Beograd: Društvo operacionih istraživača Jugoslavije, 1996.

[20] Collins WR, Miller WK, Spielman JB, Wherrey P, How Good Is Good Enough? An ethical analysis of software construction and use. Communication of ACM. 1994; 37 (1): 81-91.
[21] Bowman D, Intellectual property rights and computer software. 1996. Dostupno na: http://csrc.nist.gov/nissc/1996/papers/NISSC96/paper072_073_074/bowman.PDF. Pristupano: 10.8.2016.
[22] Butler J. Pragmatism in software copyright: Computer ssociates v. Altai. Harvard Journal of Law and Technology. 1992; 6.
[23] Easttom C, Taylor J, Computer crime, investigation, and the law, First edition, Boston: Course Technology PTR, 2011.
[24] Ottenstein K J. An algorithmic approach to the detection and prevention of plagiarism. ACM SIGSCE Bulletin, 1976; 8 (4): 30-41.
[25] Chen W, Duan C, Zheng L, Zhao Y. A hybrid method for detecting source-code plagiarism in computer programming courses. The European Conference on Education, 2013.
[26] Hage J, Rademaker P, Vugt N. A comparison of plagiarism detection tools. Technical Report UU-CS-2010-015, 2010.
[27] Mišić M, Milanović M, Protić J., Vizuelizacija rezultata detekcije plagijarizma u izvornom programskom kodu. Info M 2016; Vol. 15, Br. 57: 11-18.
[28] Roy CK, Cordy JR. A Survey on Software Clone Detection Research. Technical Report No. 2007-541, 2007.
[29] El Tahir Ali AM, Dahwa Abdulla HM, Snašsel V. Overview and comparison of plagiarism detection tools. Conference: Proceedings of the Dateso 2011: Annual International Workshop on Databases, Texts, Specifications and Objects, Pisek, Czech Republic, 2011.
[30] Marinescu D, Băicoianu A, Dimitriu A. A plagiarism detection system in computer source code. International Journal of Computer Science Research and Application. 2013; 3(1): 22-30.
[31] Đuric, Z, Gasević, D. 2013. A source code similarity system for plagiarism detection. The Computer Journal. 2013; 56 (1): 70-86.
[32] Prechelt L, Malpohl G, Phlippsen M. JPlag: Finding plagiarisms among a set of program. Technical Report 2000-1, Fakultaet fur Informatik, Universitat Karlsruhe, 2000.
[33] Dostupno na: https://jplag.ipd.kit.edu/; Pristupano: 3.7.2016.
[34] Prechelt L, Malpohl G, Phlippsen M. Finding plagiarisms among a set of programs with JPlag. Journal of Universal Computer Science. 2002; 8 (11): 1016-1038.
[35] Anbalagan S. On occurrence of plagiarism in published computer science. Master Thesis. Computer Engineering, Swedish Universitie, 2010.
[36] EC-Council. Computer Forensics: Investigation Procedures and Response, EC-Council | Press, Volume 4, 2010.
[37] Dostupno na: http://dickgrune.com/Programs/similarity_tester/ Pristupano: 7.7.2016.
[38] Gitchell D, Tran N. SIM: A utility for detecting similarity in computer programs. The proceedings of the thirtieth SIGCSE technical symposium on computer science education. 1999; 266-270.
[39] Liaqat AG, Ahmad A. Plagiarism Detection in Java Code, Degree Project. School of Computer Science, Physics and Mathematics, Linaeus University, 2011.
[40] Luke D, Johnson SL, Sreeprabha S, Varghese E. Software Plagiarism Detection Techniques: A Comparative Study. International Journal of Computer Science and Information Technologies. 2014; 5 (4): 5020-5024.

Dejan Ćapara, Software developer, Asseco SEE
Kontakt: dejan.capara@gmail.com
Oblasti interesovanja: Objektno-orijentisano programiranje, baze podataka, zaštita podataka, digitalna forenzika