2020 ‹ ŠTEVILKA 1 ‹ JAN. FEB. MAR. ‹ LETNIK XXVIII ‹ ISSN 1318-1882

01 UPORABNA INFORMATIKA

U P O R A B N A
I N F O R M A T I K A
2020 ŠTEVILKA 1 JAN/FEB/MAR LETNIK XXVIII ISSN 1318-1882

Znanstveni prispevki

Kratki znanstveni prispevki

Strokovni prispevki

U P O R A B N A
I N F O R M A T I K A

2020 ŠTEVILKA 1 JAN/FEB/MAR LETNIK XXVIII ISSN 1318-1882

Ustanovitelj in izdajatelj
Slovensko društvo INFORMATIKA
Litostrojska cesta 54, 1000 Ljubljana

Predstavnik
Niko Schlamberger

Odgovorni urednik
Saša Divjak

Tehnični urednik
Slavko Žitnik

Lektoriranje angleških izvlečkov
Marvelingua (angl.)

Oblikovanje
KOFEIN DIZAJN, d. o. o.

Prelom in tisk
Boex DTP, d. o. o., Ljubljana

Naklada
200 izvodov

Naslov uredništva
Slovensko društvo INFORMATIKA
Uredništvo revije Uporabna informatika
Litostrojska cesta 54, 1000 Ljubljana
www.uporabna-informatika.si

Revija izhaja četrtletno. Cena posamezne številke je 20,00 EUR. Letna naročnina za podjetja 85,00 EUR, za vsak nadaljnji izvod 60,00 EUR, za posameznike 35,00 EUR, za študente in seniorje 15,00 EUR. V ceno je vključen DDV.

Revija Uporabna informatika je od številke 4/VII vključena v mednarodno bazo INSPEC.

Revija Uporabna informatika je pod zaporedno številko 666 vpisana v razvid medijev, ki ga vodi Ministrstvo za kulturo RS.

Revija Uporabna informatika je vključena v Digitalno knjižnico Slovenije (dLib.si).

Vabilo avtorjem

V reviji Uporabna informatika objavljamo kakovostne izvirne članke domačih in tujih avtorjev z najširšega področja informatike v poslovanju podjetij, javni upravi in zasebnem življenju na znanstveni, strokovni in informativni ravni; še posebno spodbujamo objavo interdisciplinarnih člankov. Zato vabimo avtorje, da prispevke, ki ustrezajo omenjenim usmeritvam, pošljejo uredništvu revije po elektronski pošti na naslov ui@drustvo-informatika.si.

Avtorje prosimo, da pri pripravi prispevka upoštevajo navodila, objavljena v nadaljevanju ter na naslovu http://www.uporabna-informatika.si.

Za kakovost prispevkov skrbi mednarodni uredniški odbor. Članki so anonimno recenzirani, o objavi pa na podlagi recenzij samostojno odloča uredniški odbor. Recenzenti lahko zahtevajo, da avtorji besedilo spremenijo v skladu s priporočili in da popravljeni članek ponovno prejmejo v pregled. Uredništvo pa lahko še pred recenzijo zavrne objavo prispevka, če njegova vsebina ne ustreza vsebinski usmeritvi revije ali če članek ne ustreza kriterijem za objavo v reviji.

Pred objavo članka mora avtor podpisati izjavo o avtorstvu, s katero potrjuje originalnost članka in dovoljuje prenos materialnih avtorskih pravic. Nenaročenih prispevkov ne vračamo in ne honoriramo. Avtorji prejmejo enoletno naročnino na revijo Uporabna informatika, ki vključuje avtorski izvod revije in še nadaljnje tri zaporedne številke.

S svojim prispevkom v reviji Uporabna informatika boste prispevali k širjenju znanja na področju informatike. Želimo si čim več prispevkov z raznoliko in zanimivo tematiko in se jih že vnaprej veselimo.

Uredništvo revije

Navodila avtorjem člankov

Članke objavljamo praviloma v slovenščini, članke tujih avtorjev pa v angleščini. Besedilo naj bo jezikovno skrbno pripravljeno. Priporočamo zmernost pri uporabi tujk in – kjer je mogoče – njihovo zamenjavo s slovenskimi izrazi. V pomoč pri iskanju slovenskih ustreznic priporočamo uporabo spletnega terminološkega slovarja Slovenskega društva Informatika Islovar (www.islovar.org).

Znanstveni članek naj obsega največ 40.000 znakov, strokovni članki do 30.000 znakov, obvestila in poročila pa do 8.000 znakov.

Članek naj bo praviloma predložen v urejevalniku besedil Word (*.doc ali *.docx) v enojnem razmaku, brez posebnih znakov ali poudarjenih črk. Za ločilom na koncu stavka napravite samo en prazen prostor, pri odstavkih ne uporabljajte zamika.

Naslovu članka naj sledi za vsakega avtorja polno ime, ustanova, v kateri je zaposlen, naslov in elektronski naslov. Sledi naj povzetek v slovenščini v obsegu 8 do 10 vrstic in seznam od 5 do 8 ključnih besed, ki najbolje opredeljujejo vsebinski okvir članka. Pred povzetkom v angleščini naj bo še angleški prevod naslova, prav tako pa naj bodo dodane ključne besede v angleščini. Obratno velja v primeru predložitve članka v angleščini.

Razdelki naj bodo naslovljeni in oštevilčeni z arabskimi številkami.

Slike in tabele vključite v besedilo. Opremite jih z naslovom in oštevilčite z arabskimi številkami. Vsako sliko in tabelo razložite tudi v besedilu članka. Če v članku uporabljate slike ali tabele drugih avtorjev, navedite vir pod sliko oz. tabelo. Revijo tiskamo v črno-beli tehniki, zato barvne slike ali fotografije kot original niso primerne. Slik zaslonov ne objavljamo, razen če so nujno potrebne za razumevanje besedila. Slike, grafikoni, organizacijske sheme ipd. naj imajo belo podlago. Enačbe oštevilčite v oklepajih desno od enačbe.

V besedilu se sklicujte na navedeno literaturo skladno s pravili sistema APA navajanja bibliografskih referenc, najpogosteje torej v obliki (Novak & Kovač, 2008, str. 235). Na koncu članka navedite samo v članku uporabljeno literaturo in vire v enotnem seznamu po abecednem redu avtorjev, prav tako v skladu s pravili APA. Več o sistemu APA, katerega uporabo omogoča tudi urejevalnik besedil Word 2007, najdete na strani http://owl.english.purdue.edu/owl/resource/560/01/.

Članku dodajte kratek življenjepis vsakega avtorja v obsegu do 8 vrstic, v katerem poudarite predvsem strokovne dosežke.

Kombinirano učenje – izkušnje in rešitve

Iztok Bitenc, Borut Werber, Marko Urh
Univerza v Mariboru, Fakulteta za organizacijske vede, Kidričeva cesta 55a, 4000 Kranj
iztok.bitenc@um.si, borut.werber@um.si, marko.urh@um.si

Izvleček

Ob uvedbi kombiniranega e-učenja smo se na fakulteti predavatelji in študenti srečali z novimi izzivi. Posebej se je to pokazalo pri učenju računalniškega programiranja in podobnih predmetih, ki od študentov zahtevajo sprotno usvajanje znanja in logično kombiniranje tega znanja pri reševanju zastavljenih problemskih nalog. Tu smo zaznali povečano odsotnost študentov, plagiatorstvo in izjemno povečanje obremenitev predavateljev pri pregledovanju in ocenjevanju problemsko usmerjenih e-nalog. Rešitve smo našli na več področjih. Uvedli smo več načinov motiviranja študentov za sprotno delo. Spremenili smo vsebino e-nalog, da osnovni pregled pravilnosti oddane kode izvede računalnik. Prilagodili smo nastavitve orodja za podporo e-študiju za čim hitrejši pregled in ocenitev e--nalog. Namesto informativnih e-nalog smo izvedli e-predavanje kot videokonferenco. Z vzporednim izvajanjem predavanj in vaj smo študentom dali več časa za usvojitev novih znanj. Rezultat teh in tudi drugih ukrepov je danes za približno tretjino povečana udeležba študentov na prvem izpitnem roku.

Ključne besede: e-učenje, kombinirano učenje, Moodle, praktične izkušnje

Abstract

Blended e-learning introduced new challenges for teachers and students in our faculty. This was especially true for computer programming class and similar subjects, where students need to build up their knowledge on a regular basis and logically combine the acquired knowledge in problem-solving e-lessons. We have noticed an increased absence from classes and plagiarism, as well as an extraordinary increase of teachers' workload when grading the submitted problem-solving e-lessons. Solutions were found in different fields. We have introduced many methods to motivate students for regular study activities. Requirements for e-lessons were changed in order to use the computer as a first tool to assess the correctness of the submitted computer code. We have also changed the settings in Moodle's review module in order to streamline the grading of e-lessons. A videoconference was used instead of an e-lecture. By combining lectures and exercises in a timetable, we have given the students more time to master the new knowledge. The introduction of these and certain other measures improved the attendance of students in the first exam period by about one third.

Keywords: E-learning, blended learning, Moodle, practical experience.

1 UVOD

Po splošni definiciji je učenje proces ustvarjanja novega znanja in pridobivanja veščin skozi izkušnje. Na Fakulteti za organizacijske vede Univerze v Mariboru (FOV) nam je bolj domača definicija, ki jo ponuja Slovar slovenskega knjižnega jezika za področje šolstva: programirano učenje je individualizirano učenje po vnaprej natančno pripravljenem gradivu s pomočjo priprav za učenje ali posebnih knjig. Seveda bo potrebno to definicijo dopolniti, saj je silovit razvoj informacijske in komunikacijske tehnologije (IKT) z novimi učnimi metodami in orodji močno posegel tudi v domeno izobraževanja. Za FOV je že kmalu po prelomu tisočletja postalo zanimivo področje e-izobraževanja, saj smo imeli v preteklosti veliko število izrednih študentov, ki so jim naši predavatelji predavali na več kot 10 lokacijah po vsej Sloveniji. V e-izobraževanju smo videli možnost

zmanjšanja obremenitve predavateljev in znižanje stroškov, povezanih s predavanji na lokacijah. Tako smo v enem od laboratorijev pričeli poskusno uvajati e-izobraževanje s podporo učne platforme Moodle že leta 2004 (Leskovar, 2005). Ker so se tudi na nivoju univerze pričele odvijati aktivnosti v zvezi z uvedbo e-študija, smo z razvojem lastne rešitve počakali. Kot člani Univerze v Mariboru smo na osnovi strategije univerze leta 2011 vpeljali posebno obliko e-študija – tako imenovano *kombinirano učenje* (blended learning) – klasično poučevanje v kombinaciji z e-študijem, ki je bilo podprto na nivoju univerze z Moodle učno platformo. Približno polovica predavanj in vaj se je tako namesto v klasični obliki v predavalnicah začela izvajati izključno z računalnikom. Seveda smo ob uvedbi začeli zaznavati težave: od zamujanja pri oddaji e-nalog, povečane odsotnosti študentov na e--predavanjih in vajah, pojava plagiatorstva pri oddaji e-nalog do preobremenjenosti učiteljev zaradi pregledovanja velikega števila e-nalog. Več let smo tako iskali in izboljševali organizacijske in tudi vsebinske rešitve, ki jih v tem prispevku želimo predstaviti.

2 OPREDELITEV POJMOV

Preden opredelimo problem in predlagamo možne rešitve, je potrebno opredeliti osnovne pojme, ki se uporabljajo na tem področju. **E-študij** omogoča platformo, ki ob vsakem času in na vsakem mestu omogoča nadgradnjo znanj in spretnosti (Gowda & Suma, 2017). Zagotavlja platformo, v kateri posameznik dobi prilagojen paket, ki je povezan s ključnimi tematskimi področji, s pomočjo samovodenega procesa. To predstavlja sodelovanje tehnologije, pedagogike in akreditacije, in sicer s ciljem pripraviti privlačen koncept učenja, imenovanega e-učenje.

Glede na vključenost e-učenja v pedagoški proces ločimo dva koncepta: študij na daljavo in kombinirano učenje (»blended learning«). Medtem ko študij na daljavo običajno predstavlja študij, kjer študenti ves študij izvajajo od doma na osnovi gradiv, nalog in testov, sprva v papirnati obliki in niso nikoli fizično skupaj s predavatelji (Siemens idr., 2015)online learning, and blended learning. With the intent of informing future research and practice in the emerging discipline of digital learning, this tertiary study focuses on the history and state of distance education, and the under- standing of the large body of empirical research as captured by secondary studies (i.e., meta-analyses and systematic literature reviews, kasneje izvedeno v obliki »on-line study«, je **kombinirano učenje** mešanica klasičnega učenja v učilnici v kombinaciji z e-študijem, ki omogoča delo na daljavo z uporabo sodobnih informacijskih tehnologij (Kenney & Newcombe, 2011; Nazarenko, 2015). Z vidika usmerjenosti ločimo v učitelja usmerjeno učenje (Teacher-Centric learning) in v učence usmerjeno učenje (Learner Centric Learning) (Gowda & Suma, 2017). Medtem ko **v učitelja usmerjeno učenje** poudarja pomen pripravljenega predmetnika s postavljenimi temami, vsebinami in cilji, ki jih učitelj s svojim posredovanjem učencem poskuša doseči, se **v učence usmerjen študij** posveča učencem kot posameznikom, njihovim željam in potrebam, saj učenci dobijo aktivno vlogo v učnem procesu (skupinsko delo, medsebojno ocenjevanje, podajanje snovi s strani učencev, igranje vlog, učenje z igro …), učitelj pa je predvsem v vlogi usmerjanja in organiziranja izmenjave znanj med učenci (Gowda & Suma, 2017). V več raziskavah so proučevali razlike med e-študijem in klasičnim študijem (Daymont & Blau, 2008; Gowda & Suma, 2017; Nazarenko, 2015; William T. Kaufman, 2015). Večinoma ugotavljajo, da je osip študentov večji pri e-študiju in se kot najbolj optimalna oblika kaže kombinirano učenje. V praksi se je pokazalo, da je zelo pomembno, kaj se pri posameznem predmetu uči in kaj se pri posameznem predmetu zahteva. Ugotavlja se, da je pri e-študiju zelo pomembna komunikacija predavatelja s študentom, saj slednji hitreje izgubi interes in se izgubi, če mu pri delu ne pomaga predavatelj (Moore, 2014). Še več, ni dovolj čas komuniciranja, temveč kvaliteta komuniciranja. Osebna motivacija, oblika študijskih gradiv, učno okolje in organizacijska podpora so ključni dejavniki, ki vplivajo, da več študentov prekine e-študij (Wang, Foucar-Szocki, Griffin, O'connor, & Sceiford, 2003). V raziskavi (Anderson, 2008) so proučevani dejavniki, ki vplivajo na uspešno izvedbo e-učenja. Med slednje spadajo lastnosti študentov, učiteljev, razpoložljiva tehnologija, vsebina predmetov, podpora študentom, organiziranost inštitucije, lastnosti družbe in stroški. V tem prispevku se osredotočamo na predavatelje, študente in organiziranost predmetov.

3 ŠTUDIJ NA FOV

Na FOV imamo **seminarski način študija** – predavanja in vaje se za vsak predmet izvajajo praktično vsak

delovni dan v tednu po 3 šolske ure, običajno po dva predmeta vzporedno, pogosto tudi trije. Prvi izpitni rok je običajno 2–3 tedne po izteku predavanj in vaj. To pomeni za študente in predavatelje precej natrpan urnik. Prav tako nimamo predpisane obvezne prisotnosti na predavanjih in vajah. Z večjim številom udeležencev (do 150 študentov) se običajno srečamo v prvem letniku, ko vsi študenti obiskujejo skupni predmetnik, predmeti pa so bolj informativni. V višjih letnikih se študenti razdelijo na 3 programe in so skupine velike od 10 do 40 študentov, zahtevnost predmetov in obveznosti študentov pa se povečajo. V študijskem letu poslušajo študenti okrog 8 predmetov in 700 ur predavanj in vaj.

Programska oprema za e-študij je **Moodle** (http://moodle.org/). To je odprtokodna učna platforma, ki izobraževalcem, administratorjem in učencem zagotavlja enoten, zanesljiv in integriran sistem za ustvarjanje prilagojenih učnih okolij. Z več deset tisoč namestitvami in več kot 90 milijoni uporabnikov je največja učna platforma na svetu. Je enostavna za uporabo, preko licence GNU odprtokodna, podprta z močno skupnostjo organizacij in razvijalcev. Prevedena je v slovenščino, zanjo pa obstaja tudi več sto vtičnikov, ki dodatno razširijo njeno uporabnost (Moodle, 2019). Na FOV za podporo e-študiju pretežno uporabljamo univerzitetni strežnik, dostopen na naslovu https://estudij.um.si/, pri enem predavatelju pa tudi lokalno namestitev (https://swqlab.fov.um.si/moodle/).

Eden od uporabljenih Moodlovih vtičnikov je tudi odprtokodni spletni videokonferenčni sistem **Big Blue Button** (BBB) (http://bigbluebutton.org/), ki Moodlovi spletni učilnici doda aktivnost »videokonferenca«. BBB podpira deljenje video in avdio vsebin, naprednih predstavitev z uporabo table (kazalec, približevanje, risanje), javne in zasebne klepetalnice, deljenje zaslona in podobno. Študent spremlja predavatelja (živa slika in glas predavatelja, deljen ekran predavatelja – »tabla«, kjer predavatelj lahko kaže prosojnice ali npr. piše programsko kodo, klepetalnica, kviz) in ostale udeležence (klepetalnica, zvok, deljen ekran) v realnem času in se aktivno odziva na način, ki mu ga omogoči moderator sestanka (Big Blue Button, 2019).

4 IZKUŠNJE Z E-ŠTUDIJEM V PRAKSI

Opažene težave pri izvajanju e-študija in njihove rešitve lahko v grobem razdelimo v dve kategoriji: 1) posamezne rešitve pri specifičnih zahtevah in 2) obvladovanje velikega števila udeležencev. Pri obeh imamo podkategoriji: a) organizacijske rešitve, kjer so spremenili način izvedbe predavanj in vaj, in b) vsebinske rešitve, kjer so dejansko spremenili vsebino predmetov.

5 REŠITVE SPECIFIČNIH ZAHTEV

Predmeti, ki so se še iz časov pred uvedbo e-študija izkazali za težavne, so predmeti, kjer študente učimo osnove računalniškega programiranja v drugem letniku informacijske smeri študija. Predmeti so imeli v zgodovini poučevanja veliko različnih imen in 5 različnih programskih jezikov (trenutno sta aktualna jezika JavaScript in Java), jedro vsebine pa večinoma ostaja isto. Študente na konkretnem višjenivojskem programskem jeziku učimo osnov programiranja: od ukazov programskega jezika preko metod, ki se uporabijo za analizo problema in za izgradnjo algoritma za programsko rešitev, do dejanskega pisanja in razhroščevanja kode. Cilj predmetov je usposobiti študente, da samostojno na podlagi zahtev izdelajo pravilno delujoč program. Predmeti so med zahtevnejšimi, saj se programiranja ne da naučiti »na pamet«, ampak mora študent dejansko **razumeti** načela programiranja in obvladati pravilno uporabo gradnikov programskega jezika. Brez tega študent na izpitu, ki se vedno izvaja na računalniku, za zadano nalogo in v omejenem času enostavno ni sposoben sestaviti algoritma delovanja in na računalniku napisati delujoče programske kode.

Zahtevano znanje se pri vsakem predmetu tokom predavanj in vaj postopno nadgrajuje, kar pomeni, da mora študent za uspešno nadaljevanje vsaj dobro poznati vsebine prejšnjih predavanj in vaj. Študent, ki izpusti dan predavanj ali vaj, mora manjkajočo snov samostojno predelati do naslednjega dneva, sicer tvega, da naslednjim predavanjem ne bo mogel uspešno slediti. Ob običajnem razporedu dveh predmetov dnevno se je tako samostojno delo izkazalo kot zahtevna naloga.

Pri klasičnem podajanju predmeta pred uvedbo e-študija smo sprotno delo poskusili »vsiliti« z vmesnimi kolokviji, ki so se na koncu sešteli v oceno predmeta, vendar je na ta način uspelo zaključiti predmet le boljšim študentom oz. v povprečju manj kot tretjini. Začetne kolokvije je pisala večina študentov, izrazit osip se je pokazal, ko je snov postala zahtevnejša, npr. delo z datotekami, sploh pa pri upora-

bi eno- in dvodimenzionalnih tabel (array). Ko smo iskali razloge za osip študentov, smo pogosto dobili odgovor, da je snov postala prezahtevna in da bodo ta predmet pustili »za konec«, ko bodo imeli več časa. Iz rezultatov drugih študij izhaja, da nimamo s tem težav le pri nas (Daymont & Blau, 2008). Tak način študentskega »dela« ima pri e-študiju dodatno neprijetno posledico: povzroči dodatno obremenitev pedagoških delavcev, ki morajo v dneh pred izpitnimi roki znova nadzorovati oddaje e-nalog predmetov, ki so se dejansko že iztekli in zanje nimajo več predvidenih pedagoških ur.

E-študij smo kot člani Univerze v Mariboru vpeljali v obliki kombiniranega učenja – klasično poučevanje v kombinaciji z e-študijem. Ker nismo imeli veliko predhodnih izkušenj, je bila prva delitev časa med njima približno 50 % predavanj in vaj v e-obliki. Novonastale e-naloge so dobile dve vsebini: samostojno učenje manjših omejenih vsebin (npr. predstavitev in uporaba tekstovnih, računskih, datumskih … funkcij in metod) in utrjevanje že obdelane snovi. Primer dela naloge za utrjevanje predstavljene snovi je prikazan spodaj.

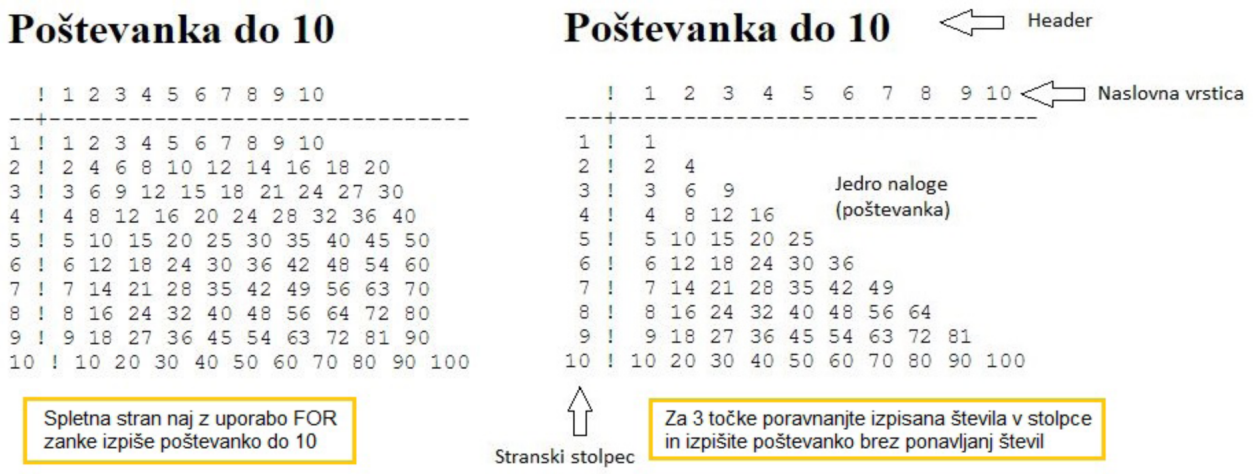
Vse e-naloge so bile zastavljene tako, da bi jih boljši študenti rešili v približno eni uri, slabši pa v treh urah. Le redki študenti so reševali e-naloge v tistih urah, ki se bile predvidene po urniku in ko smo predavatelji sedeli pred svojimi računalniki in čakali na njihova vprašanja (sinhrona e-predavanja ali vaje). Študenti so e-naloge razumeli kot asinhrona predavanja ali vaje in so e-naloge oddali enkrat v tistem dnevu, običajno v večernih urah. Zato smo predavatelji hitro prilagodili svoj način dela in čas za oddajo e-nalog temu »urniku«. Tako je zadnji čas za oddajo e-nalog (običajno 20:00) postal dejansko čas, ko smo začeli pregledovati oddane e-naloge. Delali smo torej izven rednega delovnega časa, vendar smo to kmalu uredili s pogodbami o delu na domu. Na žalost se je prav tako hitro pokazalo, da je veliko (tudi do polovice) študentov razumelo e-naloge kot dodaten prosti čas in jih niso rešili na sprejemljiv način, jih sploh niso rešili ali pa so v skrajnih primerih oddali rešitve svojih sošolcev, to je plagiate. Ker niso predelali predvidene snovi, vsi ti študenti niso imeli dobre osnove za naslednja predavanja, zato so jim sledili z opaznimi težavami.

Rešitev, kako pripraviti študente k redni oddaji e-nalog, smo najprej poiskali v prisili: oddane sprejemljive e-naloge so postale pogoj za pristop k izpitu. Še vedno pa nismo rešili težave glede prepoznega oddajanja e-nalog. Prisile, to je »zapiranja« nalog po pretečenem roku za oddajo, nismo mogli uporabiti iz več razlogov:

- že v prvem poskusu časovne omejitve oddaje se je pokazalo, da je kasneje potrebno za vsakega zamudnika posebej ročno nastaviti možnost oddaje,
- oddane e-naloge so pogoj za pristop k izpitu in zato oddaja ne sme biti onemogočena,
- za zaključek pa smo dobili še pravno mnenje, da časovno ne smemo omejiti oddaje, ker to ni v skladu z neobveznim obiskom vaj in predavanj, kar pomeni, da študent lahko zamudi oddajo.

Zato smo se namesto »prisile« odločili uporabiti »spodbudo«. Uvedli smo sistem točkovanja: za oddano e-nalogo, ki je sprejemljiva, je dobil študent točko. Vsaj eno točko za vsako e-nalogo je moral dobiti, da je lahko pristopil k izpitu. Študent je lahko dobil dodatno točko, če je v roku oddal e-nalogo. Če je zbral več kot 85 % možnih točk, je to pomenilo, da je pravočasno oddal štiri od petih e-nalog, zato se mu je končna ocena predmeta zvišala za 1 oceno. Namesto

```
/*----------------- Tekstovne funkcije in metode
Kako bi iz e-poštnega naslova ime.priimek@nekPonudnik.domena
dobili ime in priimek te osebe v dveh tekstovnih spremenljivkah?
Na primer iz začetnega e_mail="janez.novak@fov.uni-mb.si" naredite
ime = "Janez"
priimek = "Novak"
V pomoč vam je dejstvo, da je med imenom in priimkom vedno znak ".", ter da
se sklop "ime.priimek" vedno zaključi z znakom "@". V rešitvi lahko uporabite
več korakov in vmesne spremenljivke. Ne pozabite tudi na velike začetnice.
Rešitev, ki deluje samo za ime in priimek, dolga 5 znakov, ni pravilna.
*/
```



Slika 1: **Primer e-naloge.**

8, ki jo je »prislužil« na izpitu, je dobil 9. Predstavljeni model spodbude smo uporabili pri drugi generaciji študentov, ki so imeli e-študij. Izkazal se je za sicer uporabnega, vendar je bilo preveč enostavno priti do dodatne točke pri končni oceni predmeta. Zato smo ga delno spremenili.

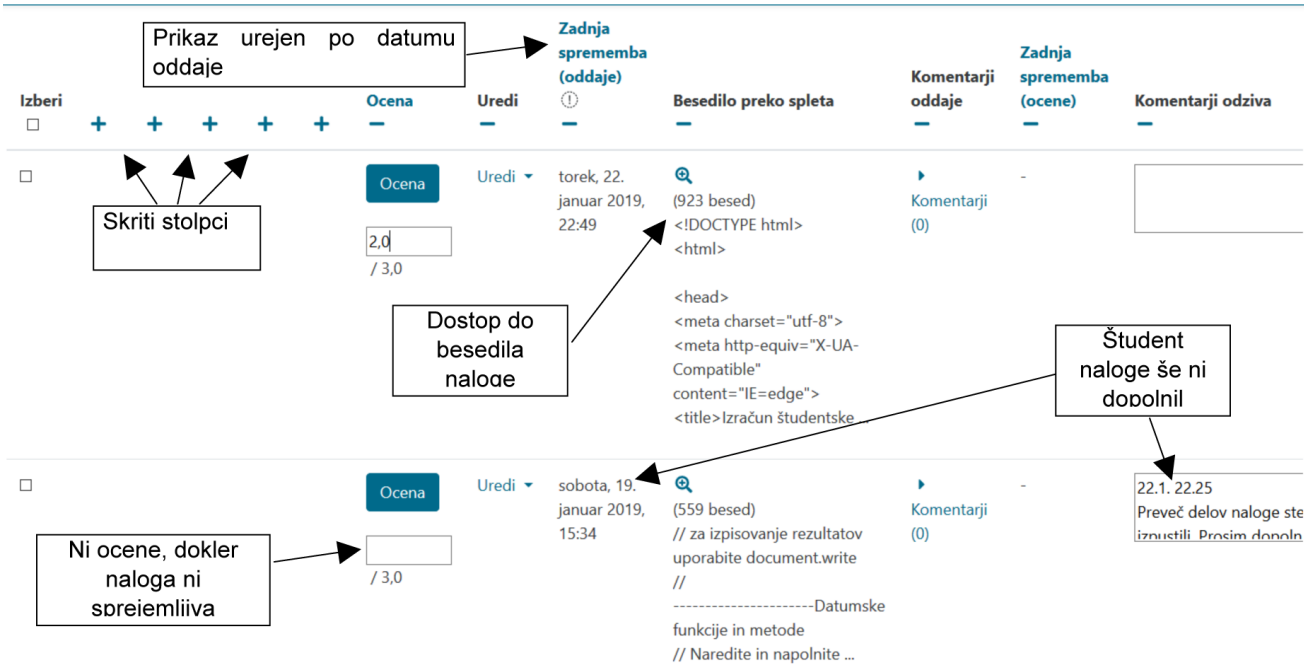
Sedaj že več let uporabljamo 3-točkovno lestvico. Prvo in obvezno točko dobi študent, ko odda sprejemljivo nalogo. Drugo dobi za pravočasno oddajo, tretjo pa, če izpolni dodatne, natančno opredeljene zahteve. Primer take naloge je prikazan na sliki 1. Osnovna naloga je, da študent napiše kodo, ki bo izpisala tabelo množenja do 10.

Precej časa smo prihranili, ko smo način pregledovanja in ocenjevanja nalog prilagodili postopku, ki ga podpira Moodle. Na začetku smo pregledovali e-naloge in dajali ocene v modulu za ocenjevanje, toda v eni od Moodlovih posodobitev prikaza oddanih e-nalog ni bilo več mogoče urediti po datumu oddanih e-nalog. To je povzročilo preveč zamude pri iskanju naknadno oddanih nalog, zato smo ta način pregledovanja in ocenjevanja hitro nadomestili z modulom *Ogled vseh oddaj*. V tem modulu vidimo in ocenimo več e-nalog naenkrat, lahko pa ga tudi do neke mere prilagodimo našim potrebam, kot je prikazano na sliki 2.

Slika 2 je bila narejena na širini ekrana 1920 točk, a smo morali prilagoditi postavitev stolpcev, da imamo na enem ekranu vidne vse podatke, ki jih potrebujemo za ocenjevanje. V stolpcih, ki so skriti, bi bili sicer prikazani: slika uporabnika (študenta), ime in priimek, številka ID, naslov e-pošte in stanje oddaje, vendar bi bil zato stolpec *Komentarji odziva*, ki ga uporabljamo za sporočila študentom, izven vidnega področja okna. Viden bi postal šele z dodatnim premikanjem levo-desno, za kar pa potrebujemo miško z dodatno funkcionalnostjo srednjega gumba.

Postopek pregledovanja in ocenjevanja e-naloge se začne, ko študent prilepi (»paste«) kodo naloge kot navadno besedilo v okvir za oddajo naloge. Prednost take oddaje pred oddajo datoteke je, da je za ocenjevalce vsebina naloge »oddaljena« samo en klik. Ocenjevalci besedilo izberemo in »prilepimo« v pripravljeno ogrodje programa, pregledamo kodo in jo poskusimo izvesti. Na podlagi rezultata nato damo ali oceno ali pa pripombe (komentarje) na nalogo. Pripombam na vrhu dodamo datum in čas nastanka pripombe, ki ga uporabimo za preverjanje, ali je študent nalogo dopolnil. Resda bi za to lahko uporabili tudi stolpec *Komentarji oddaje*, kamor se samodejno zapiše čas oddaje komentarja, toda vsebina stolpca je privzeto skrita in zahteva dodaten klik za razkritje, komentarji pa so prikazani po naraščajočem zaporedju (zadnji je vedno na dnu), kar zahteva še dodatno pomikanje vsebine okna. Držimo se tudi pravila, da študentu ne damo ocene, dokler njegova naloga ni sprejemljiva. Tako se izognemo nekaj klikom, ki so potrebni za podaljšanje roka za oddajo naloge, ki se privzeto zapre po ocenitvi.

Seveda smo uvedli tudi druge spremembe v izvedbi predavanj in vaj. Kot vsebinsko spremembo smo na vseh kontaktnih predavanjih in vajah začeli **močno poudarjati pomen rednega in sprotnega dela**. Takoj na začetku študentom povemo, da bodo morali začetniki pri programiranju za pozitivno oceno vložiti nekaj deset ur v tipkanje in preskušanje



Slika 2: **Prilagojeno okno za ocenjevanje vseh oddaj e-nalog**

delovanja programske kode, ker je to edini znani način, kako se naučiti samostojnega programiranja. V prvi polovici predavanj ali vaj jim to tudi stalno omenjamo. Prvi dan običajno predstavimo tudi najslabši scenarij, ki se pogosto zgodi študentom, ki »odložijo« predmet. Čaka jih intenziven študij v avgustu, s plačevanjem inštruktorja, udeležbo na zadnjem izpitnem roku v študijskem letu in, za okrog polovico takih študentov, ponavljanje letnika. Uspešnost takega pristopa je zaradi relativno nizkega števila udeleženih študentov (8–15 študentov) težko točno izmeriti. V zadnjih nekaj letih sta na preskusnem izpitnem roku (neuradni izpit, ki ga pišejo takoj ob zaključku predmeta, rezultat pa se upošteva pri uradnem izpitnem roku samo v primeru pozitivne ocene) običajno prisotni okrog dve tretjini študentov, kar je opazen napredek glede na eno tretjino iz obdobja pred e-študijem.

Kot organizacijsko prilagoditev smo uvedli **vzporedno izvajanje predavanj in vaj**: Po dveh ali treh dneh predavanj (spoznavanja z novo snovjo) sledita dan ali dva vaj (utrjevanje nove snovi). Tako lahko študenti, ki so npr. manjkali na predavanju ali snovi na predavanju niso dobro razumeli, na vajah še vedno usvojijo uporabo nove snovi. S tem odpadejo težave zaradi neznane snovi pri nadaljnjih predavanjih.

Predavateljem veliko časa prihrani uporaba **preprečevanja oddaje nalog** (zaklepanje), ko se predmet izteče in je bil izveden prvi izpitni rok. Ker študenti – zamudniki ne morejo več normalno oddati zahtevanih e-nalog, stopijo v stik s predavateljem ali asistentom. S tem pa ga opozorijo nase in zato asistentu ali predavatelju ni potrebno dva tedna pred vsakim izpitnim rokom več dni zapored pregledovati oddaje e-nalog za predmete, ki so se zaključili že pred več meseci. Moodle sicer lahko nastavimo, da ob vsaki oddaji e-naloge dobi predavatelj ali asistent e-poštno sporočilo, toda v praksi se je izkazalo, da to močno obremeni e-poštni predal in lastnika predala, zato smo to možnost izključili. Večino močno zamujenih e-nalog tako študenti – zamudniki opravijo s pošiljanjem nalog preko e-pošte, kar je za izvajalca bolj priročno kot globoko v Moodlovih menijih odklepati za vsakega zamudnika vsako e-nalogo posebej.

Omeniti velja tudi, da smo v študijskem letu 2017–18 za nekatere predmete **zmanjšali število ur** e-vaj in e-predavanj in povečali število kontaktnih ur. Razlog za to najbolje ilustrira primer odlične študentke, ki je dosegala nadpovprečne rezultate pri vseh predmetih, bila uvrščena na dekanovo listo najuspešnejših študentov, bila vedno prisotna na predavanjih in vajah, a je kljub temu za nalogo, planirano za eno šolsko uro, zaradi neobičajne tipkarske napake porabila

celih 5 ur samostojnega dela. Če bi isto nalogo delala v klasični predavalnici, bi ji predavatelj pomagal v nekaj sekundah. Tukaj se kaže, kako so predmeti, ki zahtevajo logično razmišljanje, povezavo na predhodna znanja in sklepanje (matematika, statistika, računalniško programiranje …), manj primerni za e-način učenja kot predmeti, ki zahtevajo pomnjenje dejstev, primerjanje, analizo, sintezo in razvrščane. Tukaj pričakujemo kritiko bralcev s pripombo, da to ne drži, saj je večina tečajev programiranja narejena v spletnih verzijah s primeri. Res je, tudi mi študentom kot dopolnilno gradivo predlagamo uporabo spletnih učilnic, kot je www3school, kjer najdejo razlage in primere osnovnih ukazov. Opazili pa smo, da študenti pristopajo k problemom na parcialen način, to pomeni, da ne preučijo snovi od začetka do konca, ampak poiščejo samo rešitev za trenuten problem, ostalo pa preskočijo. Tako pogosto v e-nalogah dobimo sicer delujoče rešitve, ki pa uporabljajo metode in kodo, ki je študentom sploh nismo predstavili. Študenti je običajno niti ne zanjo razložiti, ker so jo samo prekopirali s spleta. Seveda to postavlja pod vprašaj, ali so se s takim načinom kopiranja kode sploh kaj naučili o uporabi te kode. Težava je tudi v tem, da večina spletih učilnic brezplačno ponuja samo osnove, ki pogosto niso dovolj za naše potrebe. Za rešitve višjega nivoja pa je pri mnogih spletnih učilnicah potrebno plačilo.

Značilnost učenja programiranja je nadgradnja in povezovanje predhodnih ukazov z novimi in tudi iskanje in odpravljanje napak. Zato smo v predavanja uvedli vprašanja v obliki kratkih praktičnih nalog, v katerih študenti takoj preverijo uporabo in delovanje predstavljene snovi. Študent, ki je prisoten na predavanjih, te naloge reši tekom predavanj, odsoten študent pa mora zamujeno opraviti sam, za kar potrebuje bistveno več časa, kot če bi se udeležil predavanj in vaj. Rešitve teh kratkih nalog študent odda v spletni učilnici predmeta. Oddaja hkrati služi kot »seznam prisotnosti« za tisti dan predavanj ali vaj, saj Moodle takoj označi naloge, ki niso bile oddane pravočasno, v tem primeru do konca predavanj.

Druga prednost sprotnih nalog je, da se študent spozna z napakami v kodi in njihovim odpravljanjem. Da bi študenti čim prej spoznali večino možnih napak začetnikov, smo uvedli način predavanj s sodelovanjem študentov za katedrom. Prostovoljec (zanimivo, da se študenti sami radi javijo) rešuje nalogo za katedrom s projekcijo svojega dela na platno, predavatelj pa sledi reševanju in hkrati pomaga ostalim študentom v predavalnici. Na ta način se tudi pokažejo napake, ki jih predavatelj zaradi praktičnih izkušenj ne dela, a včasih vzamejo več deset minut predavanj, preden so skupaj odpravljene. Dodatno tudi spodbujamo boljše študente, ki so svojo nalogo že zaključili, da kot demonstratorji pomagajo sošolcem pri pisanju kode in odpravljanju napak in na tak način neposredno izvajajo učenje med seboj.

Kot dejavnik (ne)uspešne uporabe e-učenja se je pokazala tudi kultura študentov. Precej študentov pristopa k opravljanju študija po pragmatičnem načelu: do cilja z najmanj napora. Ena od »metod« tega načina študija je kopiranje rešitev v stilu »samo da nekaj oddam«. Ta dejanja ne le da povzročajo luknje v znanju študentov, ampak tudi nepotrebno dodatno delo za predavatelje, saj morajo ob vsebini preverjati tudi **plagiatorstvo**. Še posebej je to vidno pri e-nalogah, kjer vsi študenti doma rešujejo isto nalogo in imajo čas, da poiščejo rešitve prejšnjih generacij ali dobijo rešitev od boljšega študenta. Pretekle rešitve enostavno onemogočimo z delnim spreminjanjem besedila naloge (npr. namesto »izpišite najboljše tri …« zahtevamo »izpišite najboljše štiri …«), za tekoče pa imamo v okviru univerze na strežniku za e-študij nameščen vtičnik *Detektor podobnih vsebin 2.0*, ki v nekaterih primerih pomaga, pri oddaji računalniške kode pa pogosto ne. Glavna težava je, ker študenti v besedilu e-naloge že dobijo dele kode ali pripravljene podatke, detektor pa to označi kot plagiat. Tako se dejanski plagiati skrijejo v množici lažnih zadetkov.

Da bi olajšali proces učenja, študentom omogočamo uporabo povzetkov ukazov, ki vsebujejo vse ukaze, potrebne za izdelavo izpitne naloge. Povzetki niso zapisani kot rešitev, temveč kot sestavni deli kode, ki jo mora študent ustrezno združiti. Kljub opozorilom, naj takoj začnejo uporabljati povzetek ukazov, študenti z nizko motivacijo to naredijo šele tik pred izpitom. Tako jim povzetek ukazov ne pomaga veliko, ker zaradi pomanjkanja praktičnih izkušenj, ki bi jih pridobili z rednim delom, enostavno ne znajo teh delov kode smiselno uporabiti.

6 OBVLADOVANJE VELIKEGA ŠTEVILA UDELEŽENCEV

Veliko število pomeni več deset študentov, ki v istem dnevu oddajo svoje e-naloge, ki jih je treba pregledati in oceniti do naslednje izvedbe vaj ali predavanj. Tu moramo ločiti naloge glede na zvrst:

- informativne, v katerih študenti samostojno predelajo določene vsebine,
- formativne, v katerih študenti usvojijo izbrane veščine, in
- problemske, kjer morajo združiti zanje in veščine pri reševanju zastavljenega problema.

Moodle ponuja pester nabor gradnikov in orodij za razvoj **informativnih** nalog pa tudi za samodejno preverjanje kakovosti informacij, ki naj bi jih študenti z nalogo usvojili. Ta zvrst nalog tako potrebuje veliko začetnega vloženega dela: od oblikovanja vsebine za vsako nalogo, prenosa te vsebine z Moodlovimi gradniki v e-nalogo do zasnove, oblikovanja in izdelave testnih nalog in samodejnega točkovanja. Ko pa je taka e-naloga na Moodlu pripravljena, potek njenega reševanja v celoti izvaja računalnik in tako postane število udeleženih študentov nepomembno. Predavatelj ne dela več na nalogi posameznega študenta, ampak le nadzoruje uspešnost študentov in v primeru odstopanj od pričakovanih rezultatov ali velikega števila vprašanj študentov spremeni ali dopolni e-nalogo. Praktične izkušnje so pokazale, da se na tak način lahko nekatere predmete izvede skoraj v celoti, vključno z zaključnim izpitom.

Preverjanje pravilnih rešitev **formativnih** nalog je bolj zapleteno, saj so običajno odgovori opisni in je preverjanje pravilnosti odgovorov težko realizirati z razpoložljivimi Moodlovimi orodji. Najbolj zahtevne za preverjanje pa so **problemske** naloge, ki so pri učenju programiranja najbolj zastopane. V Moodlu nimamo orodij, ki bi preverjala pravilnost oddane programske kode, zato je vse delo na strani predavateljev. Ko smo prvič oblikovali e-naloge, v katerih so morali študenti samostojno reševati zastavljene probleme in oddati delujočo kodo rešitve, se je izkazalo, da že pri približno 15 oddanih e-nalogah le-teh ni bilo mogoče kakovostno pregledati v predvidenem času treh šolskih ur.

E-naloge so bile na začetku kopije nalog, ki so jih študenti delali v računalniški učilnici. Tam je predavatelj ali asistent sproti na računalniku vsakega študenta lahko spremljal njegovo delo in preverjal, ali je koda pravilna. Pri e-nalogah pa je pregledovanje potekalo na enem samem računalniku in te »paralelnosti« ni bilo. Prav tako se je pojavilo dodatno delo, ker je bilo vsako oddano e-nalogo v obliki stisnjene (»zip«) datoteke Visual Basic projekta potrebno prenesti s strežnika, prekopirati vsebino na disk predavateljevega računalnika in nato projekt odpreti z Visual Studijem. Postopek je trajal manj kot minuto, toda ko to pomnožimo z nekaj deset nalogami, se je že tako predolg čas pregledovanja nalog opazno podaljšal.

Ko smo analizirali stanje, smo ugotovili, da moramo prilagoditi e-naloge. Naloga študentov je oddati delujočo kodo, mi pa smo e-nalogam poleg zahtev, kako naj koda deluje, dodali še zahtevo, da koda naredi podrobno opisan izpis. Primer take naloge je predstavljen na sliki 1 zgoraj. Logika je enostavna – če je koda napisana pravilno, bo tudi izpis rezultatov njenega delovanja pravilen. Tako smo za osnovno »pregledovanje« uporabili računalnik – če je izpis delovanja bil tak, kot je bilo zahtevano, nam ni bilo več treba podrobno pregledovati kode vrstico za vrstico, ampak smo samo še pregledali ključne sklope kode. Če pa izpis ni bil zadosti podoben zahtevanemu, smo nalogo enostavno zavrnili, ne da bi pregledovali kodo. Na tak način smo uspeli skrajšati čas s približno 7 ur za 37 e-nalog na 3–4 ure, kar je bilo bistveno bližje predvidenim trem šolskim uram.

Zbrati več deset ljudi ob istem času na istem mestu je težava, s katero se vsak dan srečujemo v predavalnicah. Tudi pri tem smo si pomagali z uporabo sodobne IKT. Pri predmetu *Informatika in računalništvo* v prvem letniku že od leta 2004 uporabljamo lokalno verzijo Moodla (Leskovar, 2005) in od leta 2014 že prej omenjeni Moodlov vtičnik **Big Blue Button** (BBB). Namestitev vtičnika je enostavna, za delovanje pa se potrebuje spletni brskalnik z nameščenim javanskim izvedbenim okoljem JRE 1.8. Moodle je nameščen lokalno na virtualnem računalniku s štirimi procesorji, štirimi GB pomnilnika, 100 GB prostora na disku in 64-bitnim Linux Ubuntu operacijskim sistemom. Strežnik je nekaj let star HP-jev strežnik Proliant DL380 G7 z 12 jedri, 64 GB pomnilnika, 2,5 TB velikim diskovnim poljem, VMware programsko opremo in 1 GB hitro omrežno povezavo. Na tej konfiguraciji je brez vidnih težav potekalo e-predavanje z neposrednim prenosom slike in zvoka ter deljenjem ekrana s 43 hkrati prijavljenimi študenti. Strojne zahteve so sicer nižje – večina začetnih testnih sej z do 16 uporabniki je potekala na običajnem osebnem računalniku s 4-jedrnim procesorjem, 16 GB pomnilnika in 100 MB omrežne povezave. Na tej konfiguraciji smo izvedli tudi dejansko videokonferenco s 7 udeleženci.

Izvedba »videokonferenčnega« predavanja (seje) z BBB je precej podobna klasičnemu predavanju, le da ga študenti spremljajo preko računalnika. Zato je pred začetkom predavanja treba poskrbeti za dobro

osvetlitev delovnega mesta predavatelja in primerno ozadje. Administrator Moodla zažene sejo in predavatelju dodeli pravico moderatorja. Predavatelj mora pred začetkom javnega dela seje v BBB naložiti potrebne dokumente (podprta je večina oblik, kot so pdf, doc, ppt …) in v primeru, da pri izvedbi predavanja potrebuje tudi tablo, le-to nadomestiti z deljenjem dela ekrana svojega računalnika s študenti. Dobra rešitev je, da je deljeni del ekrana na istem mestu in iste velikosti, ko je okno BBB za prikazovanje naloženih dokumentov. Priporočljivo je tudi, da ima na svojem računalniku že vnaprej pripravljeno besedilo vprašanj, ki jih namerava zastaviti študentom tekom predavanja, in da že zažene aplikacije ali programe, ki jih namerava uporabiti med predavanjem. Na začetku prve seje naj bi bil poleg predavatelja prisoten tudi administrator Moodla, da pomaga študentom, ki imajo težave pri prijavi na sejo. Ker so se te težave vedno pojavile, se je prvo predavanje običajno začelo z 10- do 30-minutno zamudo.

Ko se predavanje (javni del seje) začne, predavatelj takoj prosi študente, da v BBB onemogočijo svoje mikrofone in da naj za komunikacijo uporabljajo le vgrajeno okno za klepet. S tem preprečimo zvočne motnje, ki običajno nastanejo na strani študentov. Med predavanjem predvaja prosojnice in ostale pripravljene dokumente, ki jih lahko tudi označuje ali po njih riše z vgrajenimi orodji. Ko potrebuje prikaz računalniškega ekrana (npr. za prikaz delovanja aplikacije), potegne okno te aplikacije v področje, ki ga s študenti deli na svojem ekranu. Za sprotno preverjanje razumevanja predavanja (in ugotavljanje dejanske prisotnosti študentov pred njihovimi ekrani) prilepi (»paste«) prej pripravljena besedila vprašanj in odgovorov v modul kviz, ki ga z dvema klikoma postavi v klepetalnico. Ker Moodle beleži vse dejavnosti študentov v vgrajeno bazo podatkov, lahko te podatke uporabimo za različne statistike (Leskovar in Baggia, 2015) ali celo za zbiranje točk za končno oceno.

Tak način izvedbe predavanj zahteva od predavatelja nekaj prilagajanja. Poleg nenavadnega občutka ob predavanju ekranu, mora stalno spremljati klepetalnico, da odgovori na vprašanja študentov ali dodatno pojasni snov. Paziti mora tudi na pravilno preklapljanje (prestavljanje v deljeno območje ekrana) med okni aplikacij, katerih vsebino predstavlja študentom.

Prednosti take izvedbe e-predavanja so predvsem na strani predavatelja. Na tak način lahko hitro predstavi nove vsebine, za katere še nima pripravljenih modulov na Moodlu. Ker je snov študentom predstavljena v obliki predavanja in ne e-naloge, mu ni treba preverjati odgovorov študentov na e-naloge, kar pri nekaj 10 študentih zahteva nesorazmerno veliko časa. Na študentski strani cenijo tako obliko predavanj predvsem študenti iz oddaljenih krajev in tisti, ki zjutraj težko vstanejo. Najbolje opiše njihovo mnenje anekdota, ki smo jo doživeli na enem prvih predavanj, ko je študent pri vklopljenem mikrofonu nekomu navdušeno razlagal: »Pet do osmih vstanem, si skuham kavo, vklopim računalnik in poslušam predavanje.« Navsezadnje imajo e-predavanja tudi ekološko prednost, saj prihranijo nekaj deset litrov goriva, ko študentom ni treba osebno priti na predavanja.

7 KLASIČNA V PRIMERJAVI Z E-PREDAVANJI

Sedaj po večletnih izkušnjah lahko z gotovostjo potrdimo, da je s strani izvajalcev za e-predavanja potrebno več dela in več časa kot za klasična predavanja, kar pa klub večjemu vloženemu delu ne pomeni boljšega uspeha med študenti (Daymont & Blau, 2008; William T. Kaufman, 2015). Za e-predavanja moramo najprej proučiti teorijo, izbrati neko poglavje in ga v strnjeni obliki podati v elektronski obliki kot besedilo, kodo, primere, rešitve, povezave, filme ali slike. Dodatno moramo pripraviti e-učilnico, naložiti ustrezne dokumente, predstavitve, primere in filme na ustrezna mesta in jih časovno ustrezno odpreti študentom. Sledi zbiranje rezultatov študentov in njihova ocena. Na koncu predmeta je potrebno za vsakega študenta zbrati rezultate za vsa predavanja in vaje in na osnovi tega ovrednotiti njihovo delo. Ker se skoraj nikoli ne zgodi, da bi vsi študenti v roku oddali svoje e-naloge, lahko vsak dan posebej preverjamo, če je kak od zamudnikov oddal naloge. Dodatno moramo odgovarjati na vprašanja, ki jih študenti postavljajo med izdelavo e-predavanj ali vaj. Za iskanje ene napake lahko porabimo tudi več ur kljub izkušnjam v programiranju. Dodaten problem povzroča razlika med posameznimi generacijami študentov. Redko se namreč zgodi, da druga generacija predela isto količino snovi kot predhodna, zato je potrebno že pripravljena gradiva za predavanja vsako leto prilagajati glede na hitrost sprejemanja znanj posamezne generacije. Kakor koli gledamo, je porabljen čas nesorazmeren z rezultati (Daymont & Blau, 2008) in glede na odzive študentov v anketah smo pri predmetu osnov programiranja iz 50 % prešli na 20 % e-predavanj in e-vaj.

Asinhron in sinhron način izvedbe e-predavanj in e-vaj je naslednje vprašanje, ki se nam je porajalo ob prehodu na e-študij. Sinhron način pomeni, da se v času e-predavanj izvaja neposredna komunikacija s študenti v obliki predavanj, informacij in navodil. Kljub temu da Moore ugotavlja pomen sprotne komunikacije s študenti (Moore, 2014), samo en sodelavec prakticira tak način poučevanja in odziv med študenti ni najboljši, saj s tem izničimo bistveno prednost e-študija – časovno in krajevno neodvisnost. Če mora študent točno ob določenem času biti prijavljen in poslušati predavanja preko elektronske povezave, ne vidi prednosti pred klasičnimi predavanji s tem, da tak način predavanj spremlja še slabša slišnost, slaba vidljivost, možnost prekinitev in možnost motenja v domačem okolju. Drugačen je odziv med študenti, če imajo na razpolago komunikacijo s predavateljem, ko potrebujejo nasvet ali pomoč, tudi če to ni takrat, ko sami izvajajo e-naloge. Vprašanja študentov med in po predavanjih so največji pokazatelj o tem, ali študenti spremljajo in v kolikšni meri obvladajo podano snov. Zanimivo je, da študenti postavijo več vprašanj na klasičnih predavanjih kot na e-predavanjih, kljub temu da imajo možnost zastaviti vprašanje znotraj Moodla le klik stran. Tudi če potrebujejo pomoč pri iskanju napake, se redki odločijo in pošljejo prošnjo za pomoč. Prednost klasičnih predavanj in vaj se v tem primeru pokaže tudi zaradi tega, ker ostali slišijo odgovor na postavljeno vprašanje, pri e-predavanjih pa je odgovor le za enega študenta.

Uporaba e-učenja za potrebe predštudija (Anderson, 2008) oziroma spoznavanja osnov se je pokazala kot odlična, a je žal odvisna od kulture študentov. Medtem ko je v nekaterih zahodnih državah običajno, da študenti predelajo uvodno snov, preden pridejo na predavanja, je ta kultura pri nas bolj izjema kot pravilo.

8 UGOTOVITVE

Kljub prednostim, ki jih e-študij omogoča študentom, se kaže, da v primerjavi s klasičnimi predavanji nima večjega uspeha, poleg tega pa še od študentov zahteva večjo samodisciplino in organiziranost (Daymont & Blau, 2008). Kot bolj optimalna rešitev se ponuja kombinirano učenje, ki pa samo po sebi še ne zagotavlja stimulativnega učnega okolja, zato se predlaga, da predavatelji v večji meri vključijo sodelovanje študentov in sodelovanje med njimi samimi (Kenney & Newcombe, 2011). Večinoma te ugotovitve lahko potrdimo tudi iz naših izkušenj. Predmeti, ki zahtevajo intenzivno kombiniranje predhodnih znanj in logičnega razmišljanja – tipičen predstavnik je učenje računalniškega programiranja – so manj primerni za izvedbo v obliki e-študija. Vendar tudi tu obstaja veliko rešitev, ki olajšajo delo predavateljem in povečajo uspeh študentom: predavatelji neprestano vzpodbujamo študente za samostojno, redno in sprotno delo, izboljšali smo razporeditev tega dela s primernim kombiniranjem predavanj (nova snov) in vaj (utrjevanje snovi). Predavateljem smo olajšali pregledovanje e-nalog, ko smo za osnovno oceno pravilnosti delovanja oddane kode uporabili računalnik in ko smo prilagodili orodja za ocenjevanje (Moodla) pri pregledovanju e-nalog, ali celo uporabili videokonferenco pri izvedbi e-predavanj.

Po več letih takega načina izvajanja predmetov osnove računalniškega programiranja smo z rezultati zadovoljni. Težko je sicer narediti zanesljivo primerjavo s prejšnjimi leti, ker je način dela in s tem tudi zbrani podatki precej drugačni. Če pa primerjamo stari način dela, ko je približno tretjina študentov zaključila predmet s kolokviji, s številom študentov, ki so v zadnjih nekaj letih pristopili k »preskusnemu izpitu«, je sedaj številka bistveno višja – vedno je nad 60 odstotki, v zadnjem študijskem letu smo dosegli celo 82 odstotkov. Najbolj razveseljivo pa je, da večina študentov ta preskusni izpit tudi uspešno opravi.

LITERATURA

[1] Anderson, T. (2008). *The Theory and Practice of Online Learning (Fifth Prin).* AU Press, Athabasca University. Pridobljeno od aupress@athabascau.ca

[2] Big Blue Button. *Overview (pridobljeno jul. 2019) dosegljivo na spletnem naslovu https://bigbluebutton.org/*

[3] Daymont, T., & Blau, G. (2008). *Student performance in online and traditional sections of an undergraduate management course.* Journal of Behavioral and Applied Management, 275–295. Pridobljeno od http://www.ibam.com/pubs/jbam/articles/vol9/no3/jbam_9_3_3.pdf?origin=publication_detail&sa=U&ei=8OdxU5W9I4eeyAS1-IDoCA&ved=0CCEQFjAB&usg=AFQjCNFZdwkwCvxuftcomZNP1wWZfNlNCg

[4] Gowda, R. S., & Suma, V. (2017). *A comparative analysis of traditional education system vs. e-Learning.* IEEE International Conference on Innovative Mechanisms for Industry Applications, ICIMIA 2017 - Proceedings, (Icimia), 567–571. https://doi.org/10.1109/ICIMIA.2017.7975524

[5] Kenney, J., & Newcombe, E. (2011). *Adopting a blended learning approach: Challenges encountered and lessons learned in an action research study.* Journal of Asynchronous Learning Network, 15(1), 45–57. https://doi.org/10.1016/j.sbspro.2014.01.992

[6] Leskovar, Robert. *Podpora izvajanja študija s paketom „Moodle“ - izkušnje in perspektive = Studies implementation support using the „moodle“ package - experience and perspectives.* V: KALUŽA, Jindřich (ur.). Sinergija metodologij : zbornik 24. mednarodne konference o razvoju organizacijskih znanosti, Slovenija, Portorož, 16.-18. marec 2005 = Synergy of methodologies : proceedings of the 24th International Conference on Organizational Science Development, Slovenia, Portorož, March 16-18, 2005. Kranj: Moderna organizacija. 2005, str. 550-557.

[7] Leskovar, R. & Baggia, A. (2015). *Zločin in kazen: udeležba na avditornih in videokonferenčnih predavanjih ter uspešnost reševanja nalog.* V Bernik M. in Rajkovič U. (Ur.), Vzgoja in izobraževanje v informacijski družbi - VIVID 2015, Kranj: Fakulteta za organizacijske vede, str. 287-296.

[8] Moodle, *pregled (pridobljeno jul. 2019), https://docs.moodle.org/37/en/About_Moodle*

[9] Moore, J. (2014). *Effects of online interaction and instructor presence on students' satisfaction and success with online undergraduate public relations courses.* Journalism and Mass Communication Educator, 69(3), 271–288. https://doi.org/10.1177/1077695814536398

[10] Nazarenko, A. L. (2015). *Blended Learning vs Traditional Learning: What Works?* (A Case Study Research). Procedia - Social and Behavioral Sciences, 200(October), 77–82. https://doi.org/10.1016/j.sbspro.2015.08.018

[11] Siemens, G., Skrypnyk, O., Joksimovic, S., Kovanovic, V., Dawson, S., & Gasevic, D. (2015). *The history and state of blended learning.* Preparing for the Digital University: A review of the history and current state of distance, blended, and online learning, 234. https://doi.org/10.13140/RG.2.1.3515.8483

[12] Wang, G., Foucar-Szocki, D., Griffin, O., O'connor, C., & Sceiford, E. (2003). Departure, abandonment, and dropout of e-learning: Dilemma and solutions. James Madison University, (October).

[13] William T. Kaufman. (2015). *Traditional vs. Electronic Learning Environment. Education and Human Development Master's Theses.* Paper 537. Pridobljeno od http://digitalcommons.brockport.edu/cgi/viewcontent.cgi?article=1554&context=ehd_theses.

▪

Iztok Bitenc je zaposlen kot predavatelj na Fakulteti za organizacijske vede Univerze v Mariboru na področju Informacijski sistemi, kjer sodeluje pri predmetih s področja osnov informatike, različnih programskih jezikov, razvoja spletnih rešitev in baz podatkov. Je avtor več strokovnih in znanstvenih člankov, raziskovalno pa se udejstvuje pri iskanju praktičnih rešitev z uporabo sodobne informacijske tehnologije.

▪

Borut Werber je docent na Fakulteti za organizacijske vede Univerze v Mariboru na študijskem programu Organizacija in management informacijskih sistemov. Glavna področja njegovega raziskovanja so informacijski sistemi, IKT, podkožni mikročipi in informatika v mikro-podjetjih. Aktivno sodeluje kot evalvator NAKVIS-a pri akreditacijah in reakreditacijah visokošolskih študijskih programov in institucij v Sloveniji in tujini. Kot evalvator sodeluje tudi pri notranjih evalvacijah Univerze v Mariboru. Kot recenzent aktivno sodeluje na domačih in mednarodnih konferencah ter domačih in tujih publikacijah.

▪

Marko Urh je višji predavatelj na Fakulteti za organizacijske vede Univerze v Mariboru, svojo strokovno in znanstveno pot je začel v Policiji kot razvijalec geografskih informacijskih sistemov. Glavna področja njegovega raziskovanja so informacijski sistemi, e-izobraževanje, management in razvoj kadrov. Objavil je več domačih in mednarodnih člankov in prispevkov z omenjenih področij. Aktivno sodeluje na domačih in mednarodnih konferencah ter je recenzent domačih in tujih publikacij.

Hibridni pristop za priporočanje vrstilcev univerzalne decimalne klasifikacije

Mladen Borovič, Sandi Majninger, Jani Dugonik, Marko Ferme, Milan Ojsteršek
Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko Koroška cesta 46, 2000 Maribor
mladen.borovic@um.si, sandi.majninger@um.si, jani.dugonik@um.si, marko.ferme@um.si, milan.ojstersek@um.si

Izvleček

V prispevku predstavljamo hibridni pristop za priporočanje vrstilcev univerzalne decimalne klasifikacije. S pomočjo takšnega pristopa lahko knjižničarjem omogočimo polavtomatsko določanje vrstilcev univerzalne de- cimalne klasifikacije iz vsebine že obstoječih uvrščenih gradiv. Hibridni pristop deluje na podlagi združevanja rezultata metode BM25 in naivnega Bayesovega klasifikatorja, kjer oba pristopa vrneta seznam priporočenih vrstilcev. Oba seznama združimo v končni seznam priporočil z združevalno funkcijo. V prispevku podrobneje opišemo korpus, obliko podatkov, obliko vrstilcev univerzalne decimalne klasifikacije in delovanje posamezne metode znotraj hibridnega pristopa. Podamo tudi rezultate metrik natančnosti, priklica in Fß za sezname priporočil na korpusu besedil iz nacionalne infrastrukture odprtega dostopa.

Ključne besede: digitalne knjižnice, hibridni priporočilni sistemi, programska oprema v knjižnicah, Univer- zalna decimalna klasifikacija

Abstract

In this article we present a hybrid approach to recommending the Universal Decimal Classification notation for unclassified documents. By recommending Universal Decimal Classification notation to librarians, we can enable them to semi-automatically determine the notation using already classified documents. The hybrid approach combines the BM25 method and the naive Bayes classifier, where both methods return a list of recommended notations. Both lists are merged into a final recommendation list using a custom merge function. In detail we present the Universal Decimal Classification notation structure, the corpus of documents, the inputs to our methods and the inner workings of our hybrid approach consisting of both methods. We provide the measurement results of the recommendation lists for the corpus from the National Open-Access Infrastructure in the form of precision, recall and Fß metrics.

Keywords: digital libraries, hybrid recommender systems, library software, Universal Decimal Classification

1 UVOD

Z razvojem spletnih iskalnikov sta se področji računalništva in knjižničarstva združili v interdisciplinarno področje digitalnih knjižnic, ki se ukvarja predvsem z organizacijo, skladiščenjem, obdelavo in klasifikacijo dokumentov. Predvsem klasifikacija dokumentov je raziskovalno zelo aktivno področje. Kljub temu, da je na tem področju veliko različnih metod, ne obstaja veliko metod za avtomatizirano klasificiranje po knjižničarskih klasifikatorjih, kot so univerzalna decimalna klasifikacija (UDK) [Slavic, 2004], Deweyjeva decimalna klasifikacija (DDK) [Wang, 2009] in klasifikacija Library of Congress (LCC) [Godby & Stuler, 2003], [Frank & Paynter, 2004]. Obstajajo še drugi klasifikacijski sistemi, ki so ekskluzivno namenjeni določenim jezikom (npr. v Aziji obstajajo Kitajska, Japonska in Korejska knjižničarska klasifikacija). Ne glede na sistem klasifikacije se večina gradiv po svetu še vedno klasificira ročno – bodisi zaradi nezaupanja v avtomatizirano klasifikacijo, bodisi zaradi nezadovoljivega rezultata le-te.

Problem nezaupanja v avtomatizirano klasifikacijo je potrebno s stališča knjižničarjev razumeti, saj bodo ob napačni klasifikaciji imeli dodatno delo s popravljanjem zapisov v digitalnih knjižnicah, obenem pa takšni zapisi ne bodo zlahka dostopna, saj jih uporabniki ne bodo mogli najti s trenutnimi iskalnimi postopki. V prispevku se zavedamo tega problema in v želji po zmanjšanju nezaupanja, poskušamo knjižničarjem približati avtomatizirano klasifikacijo z uvedbo priporočanja ustreznih vrstilcev klasifikacije. Ker knjižničar dobi le priporočilo, katere vrstilce naj uporabi, se lahko še vedno odloči drugače - gre torej za polavtomatsko klasifikacijo.

V prispevku opisujemo hibridni pristop priporočanja vrstilcev univerzalne decimalne klasifikacije, ki uporablja uveljavljeno iskalno metodo BM25 in naivni Bayesov klasifikator. V drugem poglavju opišemo vrste priporočilnih sistemov in uporabo le-teh v digitalnih knjižnicah. Tretje poglavje opisuje univerzalno decimalno klasifikacijo. V četrtem poglavju opišemo obliko, pripravo in obdelavo podatkov korpusa besedil iz nacionalne infrastrukture odprtega dostopa. V petem poglavju opisujemo hibridni pristop k priporočanju z uporabo metode BM25 in naivnega Bayesovega klasifikatorja. Šesto poglavje vsebuje rezultate primerjave meritev metrik natančnosti, priklica in Fβ med metodo BM25, naivnim Bayesovim klasifikatorjem in predstavljeno hibridno metodo. V sedmem poglavju podamo zaključke in nekaj idej za izboljšavo hibridne metode.

2 PRIPOROČILNI SISTEMI V DIGITALNIH KNJIŽNICAH

V zadnjih letih smo lahko opazili razmah priporočilnih sistemov na veliko področij. Dandanes se najbolj uporabljajo v spletnih iskalnikih, družbenih omrežjih in raznih multimedijskih storitvah kot so YouTube, Netflix, Spotify in Last.fm. Priporočilni sistemi za svoje delovanje v glavnem uporabljajo dva tipa filtriranja podatkov. To sta vsebinsko filtriranje (angl. content-based filtering) in sodelovalno filtriranje (angl. collaborative filtering) [Melville & Sindhwani, 2017].

Vsebinsko filtriranje podatkov uporablja opis objekta priporočanja v nestrukturirani obliki, kot je recimo besedilo, ali pa v strukturirani obliki, kjer ima objekt vnaprej znane lastnosti, po katerih definiramo filtre. Ključnega pomena je torej opis objekta priporočanja, saj ta metoda z metrikami podobnosti išče podobne objekte priporočanja. Kadar imamo opravka s podatki v strukturirani obliki, so metrike podobnosti navadno kosinusna razdalja, Jaccardov indeks in Pearsonova korelacija [Lops et al., 2011]. Nestrukturirani podatki so ponavadi podani z besedilom zato so metrike podobnosti v tem primeru omejene na metrike podobnosti, ki jih uporabljamo v procesiranju naravnega jezika. Natančneje je v tem primeru zelo pogosta uporaba utežne sheme tf -idf v kombinaciji z razvrščevalno metodo BM25.

Sodelovalno filtriranje se v nasprotju z vsebinskim filtriranjem ne osredotoča na sam opis objekta priporočanja, temveč na uporabniško interakcijo z objekti priporočanja. Za ta tip filtriranja je pomembno, ali si je uporabnik objekt priporočanja ogledal, koliko časa ga je gledal in ali je opravil kakšno pomembnejšo interakcijo s tem objektom. V primeru spletnih trgovin je to nakup izdelka, v primeru digitalnih knjižnic pa prenos dokumenta na računalnik.

Tako vsebinsko kot sodelovalno filtriranje imata svoje slabosti. Glavna slabost sodelovalnega filtriranja je problem hladnega začetka. To je situacija, v kateri se znajdemo čisto na začetku, kadar še nimamo aktivnih uporabnikov in posledično nimamo podatkov o uporabniški interakciji z objekti priporočanja. Slabost vsebinskega priporočanja je prekomerna specializacija, kjer priporočilni sistem uporabniku priporoča zgolj eno vrsto objektov priporočanja, kar pa ni vedno zaželjeno. V tem primeru se poslužimo hibridnih priporočilnih sistemov, ki združujejo dve ali več metod filtriranja v eno samo z namenom izogibanja slabostim posamezne metode. Največkrat hibridni priporočilni sistemi združujejo sodelovalno in vsebinsko filtriranje, odvisno od ciljne uporabe priporočilnega sistema pa lahko združujemo tudi več tehnik sodelovalnega filtriranja oziroma več tehnik vsebinskega filtriranja. V splošnem poznamo več načinov hibridizacije [Burke, 2002]. Z utežno hibridizacijo sestavimo oceno podobnosti iz ocen vseh vključenih metod. Pri preklopni hibridizaciji sistem preklaplja med vključenimi metodami po potrebi ali glede na situacijo. Mešana hibridizacija rezultate vključenih metod prikaže skupaj v enem seznamu priporočil. Hibridizacija s kombinacijo značilk deluje tako, da so značilke iz več virov združene in se uporabijo kot vhod v eno tehniko priporočanja. Podobno deluje hibridizacija z obogatenjem značilk, kjer se ena metoda uporabi za pridobivanje značilk, ki so vhod drugi metodi. Kaskadna hibridizacija v delovanje vnaša zaporedje uporabe različnih metod. Naza-

dnje, hibridizacija na meta ravni deluje tako, da ena metoda zgradi model, ki je vhod naslednji metodi.

V digitalnih knjižnicah se priporočilni sistemi uporabljajo predvsem v namene priporočanja dokumentov in dru- gih gradiv, ki jih digitalne knjižnice ponujajo [Bai et al., 2019]. Priporočilni sistemi opisani v [Beel et al., 2017] in [Porcel et al., 2009] so bili zasnovani specifično za uporabo v digitalnih knjižnicah z namenom, da razi- skovalcem pomagajo najti zanimive publikacije. Podobno lahko takšne priporočilne sisteme zasledimo v akademskih družbenih omrežjih, kot je na primer Mendeley [Vargas et al., 2016]. V Sloveniji obstaja hibridni priporočilni sistem, ki deluje na nacionalni infrastrukturi odprtega dostopa in navzkrižno priporoča gradiva med digitalnimi knjižnicami in repozitoriji slovenskih univerz [Ojsteršek et al., 2014]. V tem primeru gre za kaskadno hibridizacijo z metodo vsebinskega filtriranja, ki ji sledi sodelovalno filtriranje.

3 UNIVERZALNA DECIMALNA KLASIFIKACIJA

Univerzalna decimalna klasifikacija (v nadaljevanju UDK) je knjižnični klasifikacijski sistem, ki služi kot orodje za vsebinsko označevanje dokumentov in iskanje po njih. Plačljiva licenca za UDK obsega več kot 70.000 vrstilcev.Obstaja tudi zastonjska različica, ki pa je močno okrnjena na okoli 2500 vrstilcev. Z uporabo tega klasifikacijskega sistema se lahko vsakemu dokumentu določi vrstilec, ki dokument uvršča v področje. UDK sestavljajo glavne tabele in pomožne tabele, kjer glavne tabele določajo področja človeškega znanja, pomožne pa dodatne informacije o področju (npr. čas, kraj, jezik in obliko). Izraz UDK je lahko preprost ali sestavljen. V slednjem primeru se uporabijo znaki za povezovanje, ki opisujejo tip povezave med vrstilci. Tako lahko z izrazom UDK opisujemo tudi interdisciplinarne dokumente. V tabelah 1-3 so podani zgledi vrstilcev in izrazov UDK.

Tabela 1: **Vrstilci vrhnjih področij univerzalne decimalne klasifikacije.**

Vrstilec	Področja
0	Znanost in znanje. Organizacije. Informacije. Dokumentacija. Bibliotekarstvo. Institucije. Publikacije.
1	Filozofija. Psihologija.
2	Teologija. Verstva.
3	Družbene vede. Politika. Ekonomija. Pravo. Izobraževanje.
5	Matematika. Naravoslovje.
6	Uporabne znanosti. Medicina. Tehnika.
7	Umetnost. Arhitektura. Fotografija. Glasba. Šport.
8	Jezik. Književnost
9	Geografija. Biografija. Zgodovina.

Tabela 2: **Hierarhična struktura vrstilcev univerzalne decimalne klasifikacije za področje Računalništvo (004), veja Računalniške komunikacije, Računalniška omrežja (004.7).**

Vrstilec	Opis področja
004	Računalniška znanost in tehnologija. Računalništvo. Obdelava podatkov
004.7	Računalniške komunikacije. Računalniška omrežja
004.73	Omrežja glede na prostranost
004.738	Medsebojno povezovanje omrežij. Medomrežanje

Tabela 3: **Primer preprostega in sestavljenega izraza UDK. Preprost izraz vsebuje splošni privesni vrstilec za obliko (043.2). Sestavljen izraz vsebuje enostaven odnos (znak „:“), zaporedno razširitev (znak „/“), splošni privesni vrstilec za obliko (043.2) in splošni privesni vrstilec za kraj (497.4).**

Vrstilec	Izkax UDK
Preprost	519.85(043.2)
Sestavljen	336.778(043.2):336.713/.717(497.4)

Za pridobitev izraza UDK je potrebna katalogizacija oziroma zahteva knjižničarjem v primerih, ko gre za zaključna dela. Knjižničarji z uporabo geslovnika ugotovijo, katere vrstilce naj dodajo v izraz UDK tako, da v geslovnik [Zalokar, Matjaž, 2002b], [Zalokar, Matjaž, 2002a] vnesejo ključne besede oziroma predmetne oznake. V primeru zaključnih del mora avtor knjižničarjem posredovati naslov, mentorja, ključne besede, povzetek in kazalo. Knjižničarji nato iz naslova in ključnih besed pridobijo vhod za geslovnik, na podlagi povzetka, kazala in mentorja pa se dokončno odločijo za primerne vrstilce UDK. Celoten proces pridobitve izraza UDK ponavadi traja do 2 dni. Kvaliteta izraza UDK je odvisna od geslovnika in števila vrstilcev UDK, ki jih imajo knjižničarji na voljo.

4 KORPUS BESEDIL IN OBDELAVA PODATKOV

V prispevku uporabljamo korpus besedil pridobljen iz nacionalne infrastrukture odprtega dostopa [Ojsteršek et al., 2014], ki se je izvedla v letu 2013 in obsega zaključna dela in znanstvene publika- cije iz vseh slovenskih univerz. Gre za obširen korpus besedil v slovenščini, ki obsega okoli 200.000 dokumentov in je segmentiran na ključne besede, naslove, povzetke, polno besedilo in vsebuje dodatne informacije o besedilih - med njimi tudi izraze UDK. Ker vsa besedila v korpusu nacionalne infrastrukture nimajo vseh informacij na voljo, smo uporabili filtrirano podmnožico 10.000 besedil, v kateri so vsa besedila, ki imajo podatek o naslovu, ključnih besedah, polnem besedilu in izrazu UDK. V nadaljnji obdelavi podatkov smo delali s polnimi besedili, kjer smo dodatno utežili besede v naslovih in ključnih besedah.

4.1 Predobdelava besedil

Iz vseh besedil smo najprej tvorili besedne uni-, bi- in tri-grame ter izvedli vse možne permutacije med njimi. Nad besednimi n-grami smo uporabili tudi postopek lematizacije tako, da smo hkrati hranili lematizirane in nelematizirane besedne n-grame. Nato smo za to množico izračunali uteži tf in idf . Utež tf predstavlja frekvenco določenega besednega n-grama v dokumentu, utež idf pa pomembnost besednega n-grama glede na celotno zbirko dokumentov. Tako smo dobili sezname vseh možnih besednih n-gramov in njihove pojavitve v dokumentih, kot tudi število dokumentov v katerih se pojavljajo. Z enoličnim identifikatorjem dokumenta smo lahko dostopali tudi do njegovega izraza UDK in s tem povezali izraze UDK s pripadajočimi besednimi n-grami.

4.2 Razpoznavalnik izrazov UDK

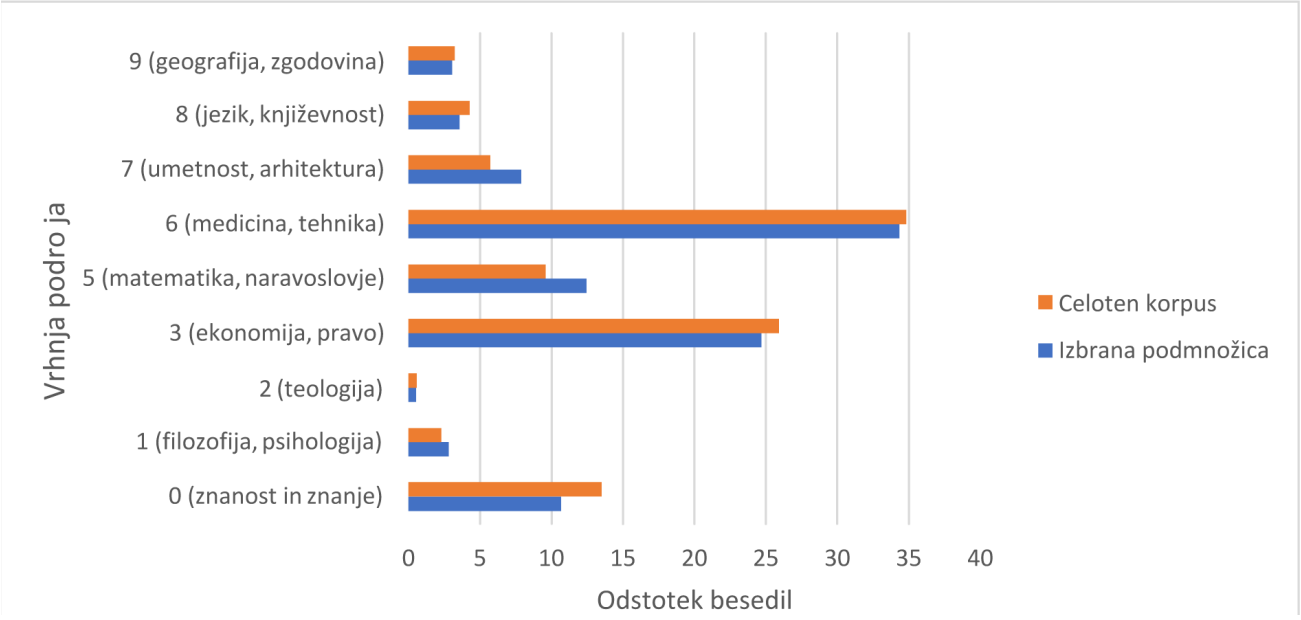
Ker je v korpusu besedil veliko takšnih, ki imajo sestavljen izraz UDK, smo zasnovali preprost razpoznavalnik izrazov UDK, ki zna iz sestavljenega izraza UDK vrniti vse vrstilce UDK. Pri tem smo upoštevali priredno in zaporedno razširitev, enostavne odnose, in podrobno delitev. Ostalih znakov za povezovanje nismo obravnavali, saj je bilo število dokumentov s temi znaki za povezovanje zanemarljivo. Prav tako nismo upoštevali splošnih privesnih vrstilcev.

Za povezavo z UDK smo uporabili brezplačno slovensko različico UDK v obliki povezanih odprtih podatkov (angl. linked open data) [UDC Consortium (UDCC), 2012]. Le-ta obsega 1445 vrstilcev UDK s slovenskim prevodom. Ta zbirka je v obliki parov (vrstilec, prevod). Zaradi omejenega števila brezplačnih vrstilcev je razpoznavanje v nekaterih primerih omejeno po globini univerzalne decimalne klasifikacije, kot je razvidno v tabeli 4.

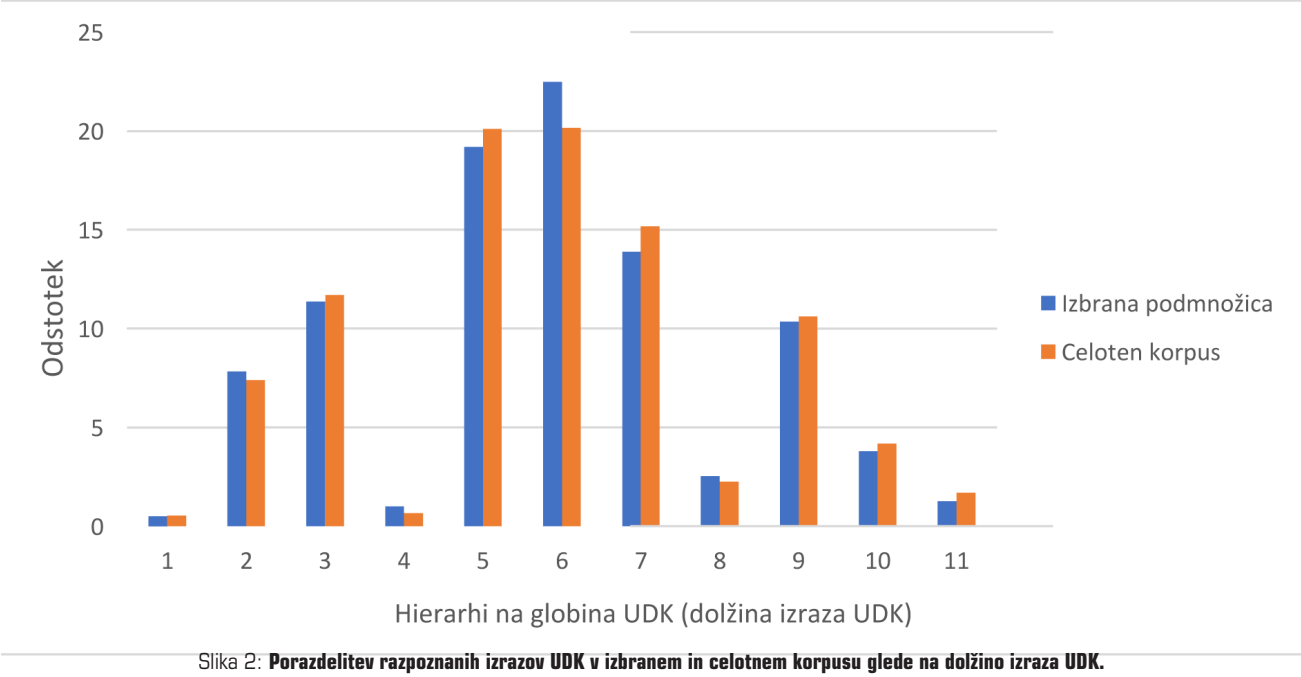
Tabela 4: **Primer delovanja razpoznavalnika izrazov UDK. Vrstilec 621.952.8 je bil razpoznan kot 621.9.**

Vhod	Izhod
(004.94:621.952.8)+658.8(043.2)	004.94 621.9 658. 003.63 8
711.4:711.1:158.937:003.63(497.4Slovenska Bistrica)(043.2)	711.4 711.1 158.937

Po obdelavi z razpoznavalnikom izrazov UDK smo preverili, kakšna je porazdelitev razpoznanih izrazov UDK v izbranem korpusu besedil. Preverili smo dolžino razpoznanih izrazov, saj dolžina izraza predstavlja globino v hierarhiji UDK in neposredno vpliva na specifičnost kategorizacije. Manjša dolžina izraza UDK pomeni splošnejšo kategorizacijo, večja dolžina izraza UDK pa specifično kategorizacijo (tabeli 1 in 2). Dolžino razpoznanega izraza UDK smo v meritvah uporabljali kot parameter. Tako smo lahko preverili, kako se uporabljene metode obnesejo na različnih nivojih specifičnosti hierarhičnih področij UDK. Slika 1 prikazuje odstotke razpoznanih izra-



Slika 1: **Porazdelitev razpoznanih izrazov UDK v izbranem in celotnem korpusu glede na vrhnja področja.**



Slika 2: **Porazdelitev razpoznanih izrazov UDK v izbranem in celotnem korpusu glede na dolžino izraza UDK.**

zov UDK v izbranem in celotnem korpusu glede na njihovo vrhnje področje. Slika 2 prikazuje odstotke razpoznanih izrazov UDK v izbranem in celotnem korpusu glede na njihovo dolžino.

5 HIBRIDNI PRISTOP K PRIPOROČANJU

V našem hibridnem pristopu uporabljamo dve metodi, ki ju uvrščamo med metode vsebinskega filtriranja. Uporabljamo metodo BM25 in naivni Bayesov klasifikator. Vhod v hibridno metodo je iskalni niz (tj. naslov, ključne besede, predmetne oznake), izhod pa je seznam najbolj ustreznih vrstilcev UDK, ki ga prikažemo knjižničarju. Ideja hibridnega pristopa je, da z obema metodama poiščemo k najbolj ustreznih vrstilcev UDK, nato pa rezultate združimo v končni seznam ustreznih vrstilcev UDK. BM25 in njene različice so že vrsto let najbolj uporabljene metode v implementaci-

jah iskalnikov (angl. full-text search) in se pojavljajo v različnih komercialnih rešitvah kot so Microsoft SQL Server, MySQL, Elasticsearch, Xapian, Solr in Lucene. Naivni Bayesov klasifikator je uveljavljena metoda na področju kategorizacije in klasifikacije besedil. V našem hibridnem pristopu ta metoda služi za uvrščanje določenih vrstilcev UDK v končni seznam priporočil, ki bi jih metoda BM25 morda izpustila.

5.1 BM25

BM25 (Best Match 25) [Robertson & Zaragoza, 2009] je metoda razvrščanja, ki omogoča razvrščanje dokumentov po podobnosti na podlagi besednih n-gramov, ki se pojavljajo v dokumentih. Začetki razvoja segajo med 1970 in 1980, ko sta avtorja začela razvijati ogrodje za pridobivanje informacij na podlagi verjetnosti. BM25 ni samo ena metoda temveč družina več metod, ki se razlikujejo po utežnih shemah in vrednostih parametrov pomembnosti za uteži. Največkrat se uporabljata uteži tf in idf . Danes obstaja veliko različic BM25, ki doprinesejo manjše izboljšave v specifičnih primerih [Trotman et al., 2014], [Lv & Zhai, 2011a], [Lv & Zhai, 2011b]. Različica BM25, ki jo uporabljamo se izračuna kot:

$$s(d, Q) = \sum_{i=1}^{||Q||} idf(q_i) \cdot \frac{tf(q_i, d) \cdot (k_1 + 1)}{tf(q_i, d) + k_1 \cdot B}, q_i \in Q, d \in D \quad (1)$$

Za enačbo 1 velja:

- tf (qi, d) je utež tf v dokumentu d za besedni n--gram qi iskalnega niza Q. Vrednost je število pojavitev besednega n-grama qi v dokumentu d.
- k1 je parameter s privzeto vrednostjo k1 = 1.2. [Manning, Christopher D. and Raghavan, Prabhakar and Schütze, H
- idf (qi) je utež idf za besedni n-gram qi. Vrednost je število pojavitev besednega n-grama qi v celotnem korpusu D. Izračun uteži idf (qi) je podan z enačbo 2

$$idf(q_i) = \log \frac{||D|| - n(q_1) + 0{,}5}{n(q_1) + 0{,}5} \quad (2)$$

  kjer je ||D|| število vseh dokumentov v korpusu D, n(qi) pa število dokumentov, ki vsebujejo besedni n-gram qi.
- B je normalizacijski faktor dan z enačbo 3

$$B = 1 - b + b \cdot \frac{l_d}{avgdl} \quad (3)$$

  kjer ld predstavlja dolžino dokumenta d, avgdl pa povprečno dolžino dokumenta glede na celoten kor- pus D. Dolžina dokumenta je izražena s številom besed v dokumentu. Parameter b ima privzeto vre- dnost b = 0.75 [Manning, Christopher D. and Raghavan, Prabhakar and Schütze, Hinrich, 2008].

Ključno vlogo imata parametra k1 in b, ki uravnavata težo uteži tf in težo dolžine dokumentov v končnem izračunu. Dolžina dokumentov se meri s številom besednih n-gramov. Parametra upoštevata dve predpostavki o značilnostih, ki se pojavljajo pri pisanju dokumentov [He & Ounis, 2003]. Predpostavka o širini vsebine dokumenta (angl. verbosity hypothesis) govori o tem, da je lahko dokument daljši zaradi uporabe nepomembnih ali redundantnih besed, medtem ko predpostavka o obsegu dokumenta (angl. scope hypothesis) govori o daljših dokumentih zaradi uporabe več besed s kontekstom, ki tvorijo vsebino dokumenta. V praksi gre za kombinacijo teh dveh predpostavk, zato potrebujemo ustrezno normalizacijo. Dolžino vsakega dokumenta lahko normaliziramo s povprečno dolžino dokumentov. Nadalje lahko to normalizacijo reguliramo s parametrom b, kot kaže enačba 3, v enačbi 1 pa vidimo, da uporabimo funkcijo normalizacije B za normalizacijo uteži tf v navezi s parametrom k1.

Parameter k1 uravnava pomembnost uteži tf , parameter b pa pomembnost dolžine dokumentov. V interesu nam je, da sestavimo takšno funkcijo, ki bo delovala najbolje na različnih dokumentih v zbirki. To pomeni, da je treba ugotoviti katere vrednosti parametrov k1 in b so najboljše za dano zbirko [He & Ounis, 2005]. Vrednosti teh dveh parametrov niso strogo definirane, navadno pa se uporabijo vrednosti k1 [1.2, 2.0] in b = [0, 1] [Manning, Christopher D. and Raghavan, Prabhakar and Schütze, Hinrich, 2008].

Nad izbranim korpusom dokumentov smo izračunali uteži tf in idf ter za vsak par dokumentov

izračunali vrednosti BM25 z upoštevanjem privzetih vrednosti za parametra k1 in b. Z metodo BM25 nato poiščemo vhodnemu besedilu najbolj podobne dokumente, vzamemo njihove izraze UDK in z razpoznavalnikom pridobimo vrstilce UDK. Vrstilce nato uredimo v seznam po frekvenci pojavljanja in vrnemo prvih k elementov tega seznama (enačbi 4 in 5).

$$R = \arg\max_{k} \{s(d_j, Q)\}, j \in [1...||D||] \quad (4)$$

$$R_x = R_{BM25} = \{udk[r]\}, \forall\ r \in R \quad (5)$$

5.2 Naivni Bayesov klasifikator

Naivni Bayesov klasifikator smo naučili nad polnim besedilom s podatkom o enoličnem identifikatorju dokumenta in pripadajočih vrstilcih UDK. Izbran korpus, opisan v poglavju 4, smo naključno razdelili na učno množico, ki je obsegala 7.000 gradiv in testno množico, ki je obsegala 3.000 gradiv. Učna in testna množica sta imeli obliko trojic (identifikator, vrstilec, besedni n-gram). Vrstilci UDK predstavljajo razrede za klasifikacijo, saj želimo klasificirati nove primerke v vrstilce UDK. Pri izračunu verjetnosti uporabljamo metodo MLE (angl. maximum likelihood estimation) in Laplaceovo (znano tudi kot Add-one) glajenje (enačbi 6 in 7). Nc predstavlja število dokumentov, ki spadajo v razred c, N je število vseh dokumentov, Tct predstavlja število pojavljanj besednega n-grama t v dokumentih iz razreda c, V predstavlja množico vseh besednih n-gramov, m pa število vseh besednih n-gramov, ki se pojavijo v vhodnem nizu. Na koncu s pomočjo naučenega modela pridobimo seznam k najbolj verjetnih vrstilcev za dan vhod (enačba 8).

$$\hat{P}(c) = \frac{N_c}{N} \quad (6)$$

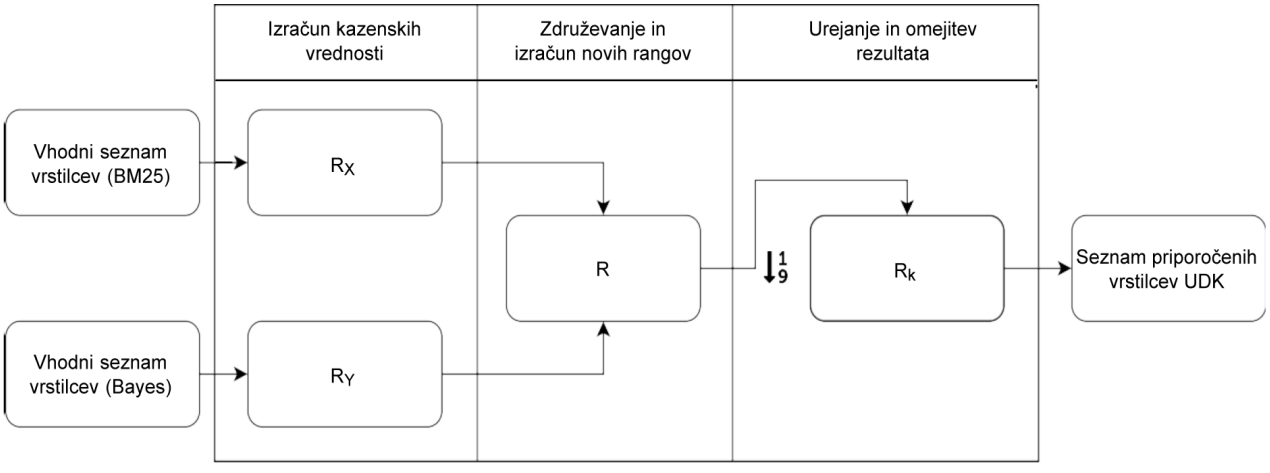
$$\hat{P}(t|c) = \frac{T_{ct} + 1}{\Sigma_{t' \in V} T_{ct'} + 1} \quad (7)$$

$$R_y = R_{Bayes} = \arg\max_{k} \{\log \hat{P}(c) + \sum_{i=1}^{m} \log \hat{P}(t_i|c)\} \quad (8)$$

5.3 Priporočanje z mešano hibridizacijo

V našem pristopu hibridnega priporočanja smo se odločili za tip mešane hibridizacije, ki združi rezultate dveh tehnik vsebinskega filtriranja (slika 3). Pristop mešane hibridizacije smo uporabili zato, ker želimo v končnem seznamu pridobiti čim več relevantnih vrstilcev UDK. ČC˘ e v skladu s pristopom mešane hibridizacije združujemo rezultate večih tehnik vsebinskega filtriranja, lahko v končnem seznamu pričakujemo vrstilce UDK, ki bi jih izpustili z uporabo zgolj ene metode vsebinskega filtriranja.

Gre torej za povečanje nabora priporočenih vrstilcev UDK v končnem seznamu priporočenih vrstilcev UDK. Seznama vrstilcev UDK, pridobljena z metodama BM25 in naivnim Bayesovim klasifikatorjem, združimo v končni seznam z združevalno funkcijo M , ki jo definiramo s psevdokodom 1.



Slika 3: **Shematika procesa priporočanja z mešano hibridizacijo.**

Algoritem 1: Združevalna funkcija M

Vhod : R_X - seznam vrstilcev UDK; rezultat prve metode vsebinskega filtriranja
Vhod : w_X - utež za enolični element v R_X; $w_X \in [0, 1]$ R
Vhod : R_Y - seznam vrstilcev UDK; rezultat druge metode vsebinskega filtriranja
Vhod : w_Y - utež za enolični element v R_Y; $w_Y \in [0, 1] \subset$ R
Vhod : k - omejitev števila elementov v izhodnem seznamu vrstilcev UDK; $k \in Z^+$
Izhod : R - seznam vrstilcev UDK, ki predstavlja združena seznama R_X in R_Y , omejen na k elementov

$R = \emptyset$
$R_k = \emptyset$
/* Element seznama je vektor r, ki vsebuje rang, vrstilec UDK in kazensko vrednost, ki je na začetku vedno enaka 0 */
$\mathbf{r} = (rank, class, penalty)$
/* Pridobimo velikosti seznamov R_X in R_Y */
$\lambda_X \leftarrow |R_X|$
$\lambda_Y \leftarrow |R_Y|$
/* Za vsak element $\mathbf{r}$ iz R_X , ki ni v R_Y izračunamo kazensko vrednost */
foreach $\mathbf{r} \in R_X \setminus R_Y$ **do**
    $\mathbf{r}.penalty \leftarrow (\lambda_X + \lambda_Y)\ w_X$
end
/* Za vsak element $\mathbf{r}$ iz R_Y , ki ni v R_X izračunamo kazensko vrednost */
foreach $\mathbf{r} \in R_Y \setminus R_X$ **do**
    $\mathbf{r}.penalty \leftarrow (\lambda_X + \lambda_Y)\ w_Y$
end
/* Združimo seznama R_X in R_Y v seznam R */
$R \leftarrow R_X \cup R_Y$
/* Za vsak element $\mathbf{r}$ iz R izračunamo nov rang na podlagi kazenskih vrednosti */
foreach $\mathbf{r} \in R$ **do**
    $\mathbf{r}.rank \leftarrow \frac{\mathbf{r}.rank + \mathbf{r}.penalty}{2}$
end
/* Seznam R uredimo naraščajoče po novih rangih */
$sort(R, \mathbf{r}.rank)$
/* Pridobimo seznam R_k, ki vsebuje prvih k elementov iz seznama R */
$R_k = \arg\max_k R$
return R_k

Ko sta na voljo seznama RX in RY , ki sta rezultat obeh metod vsebinskega filtriranja, ju je potrebno združiti z združevalno funkcijo M . Združevalna funkcija, ki jo uporabljamo, deluje na principu povprečnega ranga. V obeh seznamih iščemo enake vrstilce UDK in povprečimo njihove pozicije. Če se vrstilec pojavi v enem seznamu, v drugem pa ne, je njegov rang enak vsoti dolžin seznamov RX in RY. Takšna združevalna funkcija daje prednost tistim vrstilcem, ki so bili pridobljeni z obema metodama. Dodatno omogočimo tudi uteževanje kazenskih vrednosti na rang v primeru, da ena metoda vrne element, ki ga druga ne. Uteži kazenskih vrednosti wX in wY imata vrednosti med 0 in 1, kjer 0 ponazarja uteževanje brez vrednosti kazni, 1 pa uteževanje s polno vrednostjo kazni. Končno uteževanje lahko popolnoma spremenimo s spreminjanjem združevalne funkcije M .

6 EVALVACIJA IN REZULTATI

Merjenja uspešnosti priporočilnih sistemov se lahko lotimo na veliko načinov, saj ima vsak priporočilni sistem različen namen. Obstaja kar nekaj metod za evalvacijo priporočilnih sistemov [Pu et al., 2011], [Shani & Gunawardana, 2011], [Monti et al., 2019], [Bogaert et al., 2019], [Krauss et al., 2019]. Pred evalvacijo se moramo vprašati po rezultatu, ki ga želimo s priporočilnim sistemom doseči [Rendle et al., 2019].

V našem primeru gre za vsebinsko priporočanje, saj uporabljamo korpus besedil s katerim poskušamo najti vhodu podobne vrstilce UDK. Intuitivno lahko uporabljamo metrike kot sta natančnost in priklic, ki sta zelo znani na področjih iskalnikov in iskanju informacij [Hand & Christen, 2018], [Derczynski, 2016]. Čeprav ti dve metodi ocenjujeta uspešnost iskalnega sistema, vendarle nista zmožni oceniti uporabniške izkušnje, ki se pri priporočilnih sistemih ponavadi ocenjuje. Glavni problem knjižničarjev pri katalogiziranju je v tem, da je vrstilcev UDK veliko, hkrati pa je potrebno izbrati ustreznega. V veliki množici vrstilcev UDK je to lahko zahtevno in časovno potratno. Tako so knjižničarji zadovoljni že, če dobijo manjšo množico relevantnih vrstilcev UDK. Izmed vseh možnih vrstilcev UDK si želijo pridobiti torej samo najbolj ustrezne vrstilce UDK v pomoč, da kasneje ročno med njimi izberejo ustrezne. Zadovoljivo je tudi že, če dobijo na voljo vrhnje področje, od koder nato dalje samostojno določajo vrstilce UDK. Z vidika področja iskanja informacij gre pravzaprav za metriko priklica, ki v našem primeru meri razmerje moči množice preseka ustreznih vrstilcev UDK U in vseh vrnjenih vrstilcev UDK V , z močjo množice ustreznih vrstilcev UDK.

V našem primeru je torej metrika priklica pomembnejša od metrike natančnosti, saj gre za priporočilni sistem, ki nudi podporo pri polavtomatskem določanju vrstilcev UDK. Metrike, ki jih uporabljamo, zajemajo priklic (enačba 9), natančnost (enačba 10) in Fβ metriko (enačba 11) za vrednosti β = 1 in β = 50. Pri vrednosti β = 1 sta natančnost in priklic enakovredno uteženi, pri vrednosti β = 50 pa ima priklic 50-krat večjo težo kot natančnost.

$$R = \frac{|U| \cap |V|}{|U|} \tag{9}$$

$$P = \frac{|U| \cap |V|}{|V|} \tag{10}$$

$$F(\beta) = (1 + \beta^2) \frac{(PR)}{(\beta^2 P) + R} \tag{11}$$

Evalvacijo priporočanja vrstilcev UDK smo izvedli nad korpusom 10.000 besedil v slovenskem jeziku iz nacionalne infrastrukture odprtega dostopa, ki so imela podatek o klasifikaciji UDK. Pri tem smo izvzeli tista besedila, ki so bila v množici besedil, ki smo jih uporabili za učenje naivnega Bayesovega klasifikatorja in izračun uteži tf in idf . Meritve smo opravili za metodo BM25, naivni Bayesov klasifikator in hibridno metodo, ki združuje obe prej omenjeni metodi. Meritve smo ponovili pri različnih vrednostih za parameter kmax, ki predstavlja število vrnjenih vrstilcev. Pri tem smo se omejili na vrednosti kmax = [5, 10, 15]. V kombinaciji s parametrom kmax smo meritve ponovili tudi pri različnih vrednostih za globino hierarhije vrstilcev UDK. Globino hierarhije vrstilcev UDK udcp smo koračno po 2 znaka spreminjali na intervalu od 1 do 11 znakov. Dodatno smo v hibridni metodi spreminjali utež kazenskih vrednosti metode BM25 med 0.25 in 1 po koraku 0.25. Tabele 5, 6 in 7 vsebujejo rezultate meritev.

S hibridno metodo smo želeli povečati priklic ob predpostavki, da v našem scenariju uporabe metrika natanč- nosti ni pomembna za končnega uporabnika. Iz meritev je razvidno, da hibridna metoda v večini primerov dosega enake oziroma boljše vrednosti za metriko priklica in metriko Fβ=50 kot posamično uporabljeni metodi BM25 in Bayesov klasifikator. Opazimo, da je metoda BM25 tista, ki zagotavlja hkrati dobro natančnost in dober priklic, neodvisno od vseh preverjenih parametrov. Bayesov klasifikator je za vse preverjene vrednosti parametra kmax uporaben samo za vrhnja področja UDK (udcp = 1).

V scenariju, kadar vrnemo 5 priporočenih vrstilcev UDK (kmax = 5), hibridna metoda po metriki Fβ=50 dosega boljše vrednosti, kar je najbolj razvidno v primeru vrhnjih področij UDK (udcp = 1), za vse ostale preverjene globine hierarhije UDK pa je enakovredna metodi BM25. Največja izboljšava je pri vrhnjih področjih UDK. Kadar vrnemo 10 priporočenih vrstilcev UDK (kmax = 10) se hibridna metoda po metriki Fβ=50 znova obnese bolje kot metoda BM25. Izboljšava je vidna za vse preverjene globine hierarhije UDK, največja izboljšava pa je znova pri vrhnjih področjih UDK (udcp = 1). Kadar vrnemo 15 priporočenih vrstilcev UDK (kmax = 15), se po metriki Fβ=50 najbolje izkaže hibridna metoda na vseh globinah hierarhije UDK. Za vrhnja področja (udcp = 1) se tudi Bayesov klasifikator izkaže podobno dobro kot hibridna metoda.

Primerjali smo tudi delovanje hibridne metode ob različnih utežeh kazenskih vrednosti. V meritve in primerjavo smo vključili samo variante, kjer manj-

Tabela 5: **Rezultati meritev za uporabljene metode pri kmax = 5. Najvišje vrednosti so označene s krepko pisavo.**

kmax = 5	**Metoda**	***P***	***R***	$F_{\beta=1}$	$F_{\beta=50}$
udc_p = 1	BM25	**0.882**	0.852	**0.842**	0.852
	Bayes	0.248	0.836	0.371	0.835
	Hybrid w_{BM25} = 1.0, w_{Bayes} = 1.0	0.267	**0.891**	0.399	**0.890**
	Hybrid w_{BM25} = 0.75, w_{Bayes} = 1.0	0.267	**0.891**	0.399	**0.890**
	Hybrid w_{BM25} = 0.5, w_{Bayes} = 1.0	0.267	**0.891**	0.399	**0.890**
	Hybrid w_{BM25} = 0.25, w_{Bayes} = 1.0	0.267	**0.891**	0.399	**0.890**
udc_p = 3	BM25	**0.859**	0.908	**0.863**	0.908
	Bayes	0.097	0.343	0.147	0.343
	Hybrid w_{BM25} = 1.0, w_{Bayes} = 1.0	0.281	0.912	0.416	0.911
	Hybrid w_{BM25} = 0.75, w_{Bayes} = 1.0	0.284	0.916	0.420	0.915
	Hybrid w_{BM25} = 0.5, w_{Bayes} = 1.0	0.286	**0.921**	0.422	**0.920**
	Hybrid w_{BM25} = 0.25, w_{Bayes} = 1.0	0.286	**0.921**	0.422	**0.920**
udc_p = 5	BM25	**0.853**	**0.919**	**0.865**	**0.919**
	Bayes	0.032	0.105	0.048	0.105
	Hybrid w_{BM25} = 1.0, w_{Bayes} = 1.0	0.277	0.903	0.411	0.902
	Hybrid w_{BM25} = 0.75, w_{Bayes} = 1.0	0.286	0.918	0.423	0.917
	Hybrid w_{BM25} = 0.5, w_{Bayes} = 1.0	0.287	**0.919**	0.424	0.918
	Hybrid w_{BM25} = 0.25, w_{Bayes} = 1.0	0.287	**0.919**	0.424	0.918
udc_p = 7	BM25	**0.844**	**0.922**	**0.864**	**0.922**
	Bayes	0.049	0.154	0.072	0.154
	Hybrid w_{BM25} = 1.0, w_{Bayes} = 1.0	0.279	0.904	0.414	0.903
	Hybrid w_{BM25} = 0.75, w_{Bayes} = 1.0	0.289	**0.922**	0.426	0.921
	Hybrid w_{BM25} = 0.5, w_{Bayes} = 1.0	0.289	**0.922**	0.426	0.921
	Hybrid w_{BM25} = 0.25, w_{Bayes} = 1.0	0.289	**0.922**	0.426	0.921
udc_p = 9	BM25	**0.844**	0.922	**0.864**	0.922
	Bayes	0.051	0.161	0.075	0.161
	Hybrid w_{BM25} = 1.0, w_{Bayes} = 1.0	0.281	0.906	0.416	0.905
	Hybrid w_{BM25} = 0.75, w_{Bayes} = 1.0	0.290	**0.926**	0.427	**0.925**
	Hybrid w_{BM25} = 0.5, w_{Bayes} = 1.0	0.290	**0.926**	0.427	**0.925**
	Hybrid w_{BM25} = 0.25, w_{Bayes} = 1.0	0.290	**0.926**	0.427	**0.925**
udc_p = 11	BM25	**0.844**	0.922	**0.864**	0.922
	Bayes	0.050	0.156	0.073	0.156
	Hybrid w_{BM25} = 1.0, w_{Bayes} = 1.0	0.280	0.905	0.415	0.904
	Hybrid w_{BM25} = 0.75, w_{Bayes} = 1.0	0.290	**0.926**	0.427	**0.925**
	Hybrid w_{BM25} = 0.5, w_{Bayes} = 1.0	0.290	**0.926**	0.427	**0.925**
	Hybrid w_{BM25} = 0.25, w_{Bayes} = 1.0	0.290	**0.926**	0.427	**0.925**

šamo kazensko utež metodi BM25, ne pa tudi Bayesovemu klasifikatorju. Tako smo se odločili zato, ker manjšanje kazenskih uteži Bayesovemu klasifikatorju ne vodi v izboljšanje rezultatov metrik natančnosti, priklica, Fβ=1 in Fβ=50. Iz rezultatov meritev vidimo, da se manjšanje kazenskih uteži metodi BM25 splača vsaj do polovične vrednosti kazenske uteži (wBM25 = 0.5) za 5 vrnjenih zadetkov in vsaj do tričetrt vrednosti kazenske uteži (wBM25 = 0.75) za 10 in 15 vrnjenih zadetkov.

Tabela 6: **Rezultati meritev za uporabljene metode pri kmax = 10. Najvišje vrednosti so označene s krepko pisavo.**

kmax = 10	**Metoda**	***P***	***R***	$F_{\beta=1}$	$F_{\beta=50}$
$udc_p = 1$	BM25	**0.880**	0.852	**0.840**	0.852
	Bayes	0.146	0.902	0.245	0.900
	Hybrid $w_{BM25} = 1.0$, $w_{Bayes} = 1.0$	0.147	**0.906**	0.247	**0.904**
	Hybrid $w_{BM25} = 0.75$, $w_{Bayes} = 1.0$	0.147	**0.906**	0.247	**0.904**
	Hybrid $w_{BM25} = 0.5$, $w_{Bayes} = 1.0$	0.147	**0.906**	0.247	**0.904**
	Hybrid $w_{BM25} = 0.25$, $w_{Bayes} = 1.0$	0.147	**0.906**	0.247	**0.904**
$udc_p = 3$	BM25	**0.855**	0.914	**0.859**	0.914
	Bayes	0.062	0.439	0.107	0.438
	Hybrid $w_{BM25} = 1.0$, $w_{Bayes} = 1.0$	0.134	0.921	0.242	0.919
	Hybrid $w_{BM25} = 0.75$, $w_{Bayes} = 1.0$	0.144	0.923	0.243	0.921
	Hybrid $w_{BM25} = 0.5$, $w_{Bayes} = 1.0$	0.144	**0.927**	0.244	**0.925**
	Hybrid $w_{BM25} = 0.25$, $w_{Bayes} = 1.0$	0.144	**0.927**	0.244	**0.925**
$udc_p = 5$	BM25	**0.848**	0.920	**0.859**	0.920
	Bayes	0.032	0.212	0.055	0.212
	Hybrid $w_{BM25} = 1.0$, $w_{Bayes} = 1.0$	0.144	0.925	0.411	0.902
	Hybrid $w_{BM25} = 0.75$, $w_{Bayes} = 1.0$	0.145	**0.926**	0.245	0.923
	Hybrid $w_{BM25} = 0.5$, $w_{Bayes} = 1.0$	0.145	**0.926**	0.245	**0.924**
	Hybrid $w_{BM25} = 0.25$, $w_{Bayes} = 1.0$	0.145	**0.926**	0.245	**0.924**
$udc_p = 7$	BM25	**0.841**	0.925	**0.859**	0.925
	Bayes	0.035	0.217	0.059	0.217
	Hybrid $w_{BM25} = 1.0$, $w_{Bayes} = 1.0$	0.145	0.930	0.246	0.928
	Hybrid $w_{BM25} = 0.75$, $w_{Bayes} = 1.0$	0.146	0.932	0.247	0.930
	Hybrid $w_{BM25} = 0.5$, $w_{Bayes} = 1.0$	0.146	**0.933**	0.248	**0.931**
	Hybrid $w_{BM25} = 0.25$, $w_{Bayes} = 1.0$	0.146	**0.933**	0.248	**0.931**
$udc_p = 9$	BM25	**0.840**	0.925	**0.859**	0.925
	Bayes	0.033	0.209	0.056	0.209
	Hybrid $w_{BM25} = 1.0$, $w_{Bayes} = 1.0$	0.146	0.932	0.247	0.905
	Hybrid $w_{BM25} = 0.75$, $w_{Bayes} = 1.0$	0.128	0.824	0.217	0.822
	Hybrid $w_{BM25} = 0.5$, $w_{Bayes} = 1.0$	0.146	**0.933**	0.248	**0.931**
	Hybrid $w_{BM25} = 0.25$, $w_{Bayes} = 1.0$	0.146	**0.933**	0.248	**0.931**
$udc_p = 11$	BM25	**0.840**	0.925	**0.858**	0.925
	Bayes	0.032	0.203	0.055	0.203
	Hybrid $w_{BM25} = 1.0$, $w_{Bayes} = 1.0$	0.146	0.932	0.247	0.930
	Hybrid $w_{BM25} = 0.75$, $w_{Bayes} = 1.0$	0.146	0.932	0.247	0.930
	Hybrid $w_{BM25} = 0.5$, $w_{Bayes} = 1.0$	0.146	**0.933**	0.248	**0.931**
	Hybrid $w_{BM25} = 0.25$, $w_{Bayes} = 1.0$	0.146	**0.933**	0.248	**0.931**

Glede na porazdelitev razpoznanih izrazov UDK na hierarhično globino UDK (slika 2) smo ugotovili, da v primeru manjšega števila vrnjenih zadetkov ni bistvene razlike med uporabo BM25 in predlagane hibridne metode, kadar govorimo o odstotkovno največji pokritosti izbranega korpusa besedil, ki nastopi pri vrednostih parametra udcp = 5 in udcp = 7 ter metrikah priklica in Fβ=50. V splošnem smo ugotovili, da so vrednosti izbranih metrik približno enake za hierarhično globino UDK nad 7 znakov. Kadar pa se

Tabela 7: **Rezultati meritev za uporabljene metode pri kmax = 15. Najvišje vrednosti so označene s krepko pisavo.**

kmax = 15	**Metoda**	***P***	***R***	$F_{\beta=1}$	$F_{\beta=50}$
udc_p = 1	BM25	**0.880**	0.852	**0.840**	0.852
	Bayes	0.146	0.902	0.245	0.900
	Hybrid w_{BM25} = 1.0, w_{Bayes} = 1.0	0.146	**0.906**	0.247	**0.904**
	Hybrid w_{BM25} = 0.75, w_{Bayes} = 1.0	0.146	**0.906**	0.247	**0.904**
	Hybrid w_{BM25} = 0.5, w_{Bayes} = 1.0	0.146	**0.906**	0.247	**0.904**
	Hybrid w_{BM25} = 0.25, w_{Bayes} = 1.0	0.146	**0.906**	0.247	**0.904**
udc_p = 3	BM25	**0.854**	0.916	**0.857**	0.916
	Bayes	0.047	0.485	0.084	0.483
	Hybrid w_{BM25} = 1.0, w_{Bayes} = 1.0	0.096	0.930	0.172	0.927
	Hybrid w_{BM25} = 0.75, w_{Bayes} = 1.0	0.097	**0.931**	0.172	**0.928**
	Hybrid w_{BM25} = 0.5, w_{Bayes} = 1.0	0.097	**0.931**	0.172	**0.928**
	Hybrid w_{BM25} = 0.25, w_{Bayes} = 1.0	0.097	**0.931**	0.172	**0.928**
udc_p = 5	BM25	**0.846**	0.921	**0.857**	0.921
	Bayes	0.038	0.361	0.067	0.360
	Hybrid w_{BM25} = 1.0, w_{Bayes} = 1.0	0.097	0.936	0.174	0.933
	Hybrid w_{BM25} = 0.75, w_{Bayes} = 1.0	0.098	**0.938**	0.174	**0.935**
	Hybrid w_{BM25} = 0.5, w_{Bayes} = 1.0	0.098	**0.938**	0.174	**0.935**
	Hybrid w_{BM25} = 0.25, w_{Bayes} = 1.0	0.098	**0.938**	0.174	**0.935**
udc_p = 7	BM25	**0.839**	0.929	**0.857**	0.929
	Bayes	0.025	0.231	0.044	0.230
	Hybrid w_{BM25} = 1.0, w_{Bayes} = 1.0	0.098	0.936	0.174	0.933
	Hybrid w_{BM25} = 0.75, w_{Bayes} = 1.0	0.098	0.935	0.174	0.932
	Hybrid w_{BM25} = 0.5, w_{Bayes} = 1.0	0.098	**0.939**	0.175	**0.936**
	Hybrid w_{BM25} = 0.25, w_{Bayes} = 1.0	0.098	**0.939**	0.175	**0.936**
udc_p = 9	BM25	**0.838**	0.925	**0.856**	0.925
	Bayes	0.024	0.223	0.042	0.222
	Hybrid w_{BM25} = 1.0, w_{Bayes} = 1.0	0.098	**0.936**	0.174	**0.933**
	Hybrid w_{BM25} = 0.75, w_{Bayes} = 1.0	0.098	**0.936**	0.174	**0.933**
	Hybrid w_{BM25} = 0.5, w_{Bayes} = 1.0	0.098	**0.936**	0.174	**0.933**
	Hybrid w_{BM25} = 0.25, w_{Bayes} = 1.0	0.098	**0.936**	0.174	**0.933**
udc_p = 11	BM25	**0.838**	0.925	**0.856**	0.925
	Bayes	0.023	0.217	0.041	0.216
	Hybrid w_{BM25} = 1.0, w_{Bayes} = 1.0	0.098	**0.936**	0.174	**0.933**
	Hybrid w_{BM25} = 0.75, w_{Bayes} = 1.0	0.098	**0.936**	0.174	**0.933**
	Hybrid w_{BM25} = 0.5, w_{Bayes} = 1.0	0.098	**0.936**	0.174	**0.933**
	Hybrid w_{BM25} = 0.25, w_{Bayes} = 1.0	0.098	**0.936**	0.174	**0.933**

število vrnjenih zadetkov poveča, predlagana hibridna metoda konstantno vrača višje vrednosti izbranih metrik neodvisno od izbrane hierarhične globine UDK. Zaključujemo torej, da je uporaba predlagane hibridne metode ustrezna za polavtomatsko določanje vrstilcev UDK v obliki priporočilnega sistema, kjer knjižničarji dobijo predlagane vrstilce UDK na podlagi vhodnega besedila, med katerimi nato ročno izberejo ustrezne.

7 SKLEP

V članku smo predstavili hibridni pristop za priporočanje vrstilcev univerzalne decimalne klasifikacije. Opisali smo izbran korpus in predobdelavo besedil za uporabo v predlagani hibridni metodi. Prikazali smo kako z mešano hibridizacijo uporabimo metodi BM25 in naivni Bayesov klasifikator ter opisali preprosto združevalno funkcijo, ki oblikuje končni rezultat. Izvedli smo evalvacijo hibridne metode, metode BM25 in naivnega Bayesovega klasifikatorja, kjer smo ugotovili, da se hibridna metoda obnese bolje za metriki priklica in Fβ=50, ki sta bolj relevantni kot metrika natančnosti za scenarij uporabe sistema kot orodja za knjižničarje.

Predstavljen hibridni pristop lahko spreminjamo na več načinov in na več mestih. Ena izmed možnosti izboljšave je uporaba licenčne različice UDK vrstilcev, saj bi tako uspešno razpoznali večji delež izrazov UDK, še posebej na višji hierarhični globini UDK. Prav tako bi lahko izvedli optimizacijo metode BM25 za korpus, ki smo ga uporabljali, kjer bi z optimiziranjem parametrov k1 in b lahko iskali manjše izboljšave. Podobno bi lahko optimizirali vrednosti uteži kazenskih vrednosti. Hibridni pristop je vedno možno izboljšati s spreminjanjem združevalne funkcije M glede na potrebe končnega uporabnika ali pa z različnim načinom hibridizacije. Pri tem bi bila zanimiva predvsem utežni in kaskadni tip hibridizacije. Predstavljen hibridni pristop je prav tako ustrezen za uporabo pri določanju kandidatov dokumentov za podrobnejše preverjanje v sistemu za detekcijo podobnih vsebin. Nazadnje bi bilo zanimivo videti tudi, kako se na tem področju obnesejo nevronske mreže s povratno zanko, ki so v zadnjem obdobju zelo napredovale na področjih besedilnega rudarjenja in obdelave naravnega jezika.

LITERATURA

[1] Bai, X., Wang, M., Lee, I., Yang, Z., Kong, X., & Xia, F. (2019). *Scientific Paper Recommendation: A Survey*. IEEE Access, 7, 9324–9339.

[2] Beel, J., Aizawa, A., Breitinger, C., & Gipp, B. (2017). Mr. DLib: *Recommendations-as-a-Service (RaaS) for Academia*. In 2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL) (pp. 1–2).

[3] Bogaert, M., Lootens, J., den Poel, D. V., & Ballings, M. (2019). *Evaluating multi-label classifiers and recommender systems in the financial service sector.* European Journal of Operational Research, 279(2), 620 – 634.

[4] Burke, R. (2002). *Hybrid Recommender Systems: Survey and Experiments.* User Modeling and User-Adapted Interaction, 12(4), 331–370.

[5] Derczynski, L. (2016). *Complementarity, F-score, and NLP Evaluation. In Proceedings of the Tenth International Conference on Language Resources and Evaluation* (LREC 2016) (pp. 261–266). Portorož, Slovenia: European Language Resources Association (ELRA).

[6] Frank, E. & Paynter, G. W. (2004). *Predicting Library of Congress Classifications from Library of Congress Subject Headings.* J. Am. Soc. Inf. Sci. Technol., 55(3), 214–227.

[7] Godby, C. J. & Stuler, J. (2003). *The Library of Congress Classification as a Knowledge Base for Automatic Subject Categorization.* In Subject Retrieval in a Networked Environment: Proceedings of the IFLA Satellite Meeting held in Dublin, OH,14-16 August 2001 and sponsored by the IFLA Classification and Indexing Section, the IFLA Information Technology Section and OCLC (pp. 163–169).

[8] Hand, D. & Christen, P. (2018). *A note on using the F-measure for evaluating record linkage algorithms.* Statistics and Computing, 28(3), 539–547.

[9] He, B. & Ounis, I. (2003). *A Study of Parameter Tuning for Term Frequency Normalization.* In Proceedings of the Twelfth International Conference on Information and Knowledge Management, CIKM '03 (pp. 10–16). New York, NY, USA: ACM.

[10] He, B. & Ounis, I. (2005). Term Frequency Normalisation Tuning for BM25 and DFR Models. In D. E. Losada & J. M. Fernández-Luna (Eds.), Advances in Information Retrieval (pp. 200–214). Berlin, Heidelberg: Springer Berlin Heidelberg.

[11] Krauss, C., Merceron, A., & Arbanowski, S. (2019). *The Timeliness Deviation: A Novel Approach to Evaluate Educational Recommender Systems for Closed-Courses.* In Proceedings of the 9th International Conference on Learning Analytics & Knowledge, LAK19 (pp. 195–204). New York, NY, USA: ACM.

[12] Lops, P., de Gemmis, M., & Semeraro, G. (2011). *Content-based Recommender Systems: State of the Art and Trends,* (pp. 73–105). Springer US: Boston, MA.

[13] Lv, Y. & Zhai, C. (2011a). *Adaptive Term Frequency Normalization for BM25.* In Proceedings of the 20th ACM International Conference on Information and Knowledge Management, CIKM '11 (pp. 1985–1988). New York, NY, USA: ACM.

[14] Lv, Y. & Zhai, C. (2011b). *Lower-bounding Term Frequency Normalization. In Proceedings of the 20th ACM International Conference on Information and Knowledge Management,* CIKM '11 (pp. 7–16). New York, NY, USA: ACM.

[15] Manning, Christopher D. and Raghavan, Prabhakar and Schütze, Hinrich (2008). *Introduction to Information Retrieval.* New York, NY, USA: Cambridge University Press.

[16] Melville, P. & Sindhwani, V. (2017). *Recommender Systems*, (pp. 1056–1066). Springer US: Boston, MA.

[17] Monti, D., Palumbo, E., Rizzo, G., & Morisio, M. (2019). *Sequeval: An Offline Evaluation Framework for Sequence-Based Recommender Systems*. Information, 10, 174.

[18] Ojsteršek, M., Brezovnik, J., Kotar, M., Ferme, M., Hrovat, G., Bregant, A., & Borovič, M. (2014). *Establishing of a Slovenian open access infrastructure: a technical point of view.* Program, 48(4), 394–412.

[19] Porcel, C., Moreno, J., & Herrera-Viedma, E. (2009). *A multi-disciplinar recommender system to advice research resources in University Digital Libraries*. Expert Systems with Applications, 36(10), 12520–12528.

[20] Pu, P., Chen, L., & Hu, R. (2011). *A User-centric Evaluation Framework for Recommender Systems.* In Proceedings of the Fifth ACM Conference on Recommender Systems, RecSys '11 (pp. 157–164). New York, NY, USA: ACM.

[21] Rendle, S., Zhang, L., & Koren, Y. (2019). *On the Difficulty of Evaluating Baselines: A Study on Recommender Systems.* ArXiv, abs/1905.01395.

[22] Robertson, S. & Zaragoza, H. (2009). *The Probabilistic Relevance Framework. now.*
[23] Shani, G. & Gunawardana, A. (2011). *Evaluating Recommendation Systems*, (pp. 257–297). Springer US: Boston, MA.
[24] Slavic, A. (2004). *UDC implementation: From library shelves to a structured indexing language.* In International Cataloguing and Bibliographic Control., volume 33.3 (pp. 60–65).
[25] Trotman, A., Puurula, A., & Burgess, B. (2014). *Improvements to BM25 and Language Models Examined.* In Proceedings of the 2014 Australasian Document Computing Symposium, ADCS '14 (pp. 58:58–58:65). New York, NY, USA: ACM.
[26] UDC Consortium (UDCC) (2012). *Multilingual Universal Decimal Classifi- cation Summary* (UDCC Publication No. 088).
[27] Vargas, S., Hristakeva, M., & Jack, K. (2016). *Mendeley: Recommendations for Researchers.* In RecSys '16 Proceedings of the 10th ACM Conference on Recommender Systems (pp. 365–365). Boston, MA, USA.
[28] Wang, J. (2009). *An extensive study on automated Dewey Decimal Classification.* Journal of the American Society for Information Science and Technology, 60(11), 2269–2286.
[29] Zalokar, Matjaž (2002a). *Spletni splošni slovenski geslovnik*. http://old.nuk.uni-lj.si/ssg/geslovnik.html.
[30] Zalokar, Matjaž (2002b). *Splošni slovenski geslovnik.* Organizacija znanja, 7, 3–4.

▪

Mladen Borovič je doktorski študent in asistent na Fakulteti za elektrotehniko, računalništvo in informatiko na Univerzi v Mariboru. Njegovo raziskovalno delo obsega področja priporočilnih sistemov, iskalnih sistemov, porazdeljenih računalniških sistemov, odkrivanja podobnih vsebin, besedilnega rudarjenja in obdelave naravnega jezika. Še posebej se ukvarja s hibridnimi priporočilnimi sistemi in uporabo metod umetne inteligence v besedilnem rudarjenju.

▪

Sandi Majninger je doktorski študent in asistent na Fakulteti za elektrotehniko, računalništvo in informatiko na Univerzi v Mariboru. Raziskovalno je aktiven na področju obdelave naravnega jezika, odkrivanja podobnih vsebin ter ugotavljanju pomena iz besedil. Med drugim se ukvarja tudi z avtomatskim ocenjevanjem pomenske pravilnosti odgovorov na vprašanja odprtega tipa in avtomatskim ocenjevanjem daljših pisnih sestavkov ter esejev.

▪

Jani Dugonik je doktorski študent in asistent na Fakulteti za elektrotehniko, računalništvo in informatiko. Nje- gova raziskovalna področja vključujejo evolucijsko računanje, optimizacijske metode, procesiranje naravnega jezika in globoko učenje.
Marko Ferme je raziskovalec na Fakulteti za elektrotehniko, računalništvo in informatiko na Univerzi v Mariboru. Njegova raziskovalna področja obsegajo procesiranje naravnega jezika, sisteme za odgovarjanje na vprašanja v naravnem jeziku, ontologije in semantični splet, aktiven pa je tudi na več raziskovalnih in komercialnih projektih na področju digitalnih knjižnic.ziskovalnih projektih s področja strateškega planiranja, metodologij razvoja informacijskih sistemov, uporabe inteligentnih sistemov, avtomatizacije poslovnih procesov in obvladovanja ter porazdelitve velike količine podatkov.

▪

Milan Ojsteršek je raziskovalec na Fakulteti za elektrotehniko, računalništvo in informatiko na Univerzi v Mariboru. Njegova raziskovalna področja zajemajo heterogene računalniške sisteme, digitalne knjižnice, semantični splet in storitveno usmerjene arhitekture.Marko Ferme je raziskovalec na Fakulteti za elektrotehniko, računalništvo in informatiko na Univerzi v Mariboru. Njegova raziskovalna področja obsegajo procesiranje naravnega jezika, sisteme za odgovarjanje na vprašanja v naravnem jeziku, ontologije in semantični splet, aktiven pa je tudi na več raziskovalnih in komercialnih projektih na področju digitalnih knjižnic.ziskovalnih projektih s področja strateškega planiranja, metodologij razvoja informacijskih sistemov, uporabe inteligentnih sistemov, avtomatizacije poslovnih procesov in obvladovanja ter porazdelitve velike količine podatkov.

Decentralizirano in odporno dinamično posodabljanje mikrostoritev

Jan Meznarič, Matjaž B. Jurič
Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, Večna pot 113, 1000 Ljubljana
{jan.meznaric, matjaz.juric}@fri.uni-lj.si

Izvleček

Za zagotavljanje neprekinjenega delovanja aplikacije je potrebno vpeljati dinamično posodabljanje programske opreme, ki med procesom posodobitve ne povzroča izpada storitev. V arhitekturi mikrostoritev za dinamično posodabljanje običajno uporabimo centralizirana orodja za orkestracijo vsebnikov. Članek opisuje izsledke raziskovalnega dela, v katerem razvijamo decentralizirano metodo za dinamično posodabljanje mikrostoritev. Predlagana metoda definira koordinatorje posodobitve za decentralizirano upravljanje procesa posodobitve, ki z mehanizmi za izboljšanje odpornosti na napake izvedejo evalvacijo nove verzije mikrostoritve. Z evalvacijo preprečimo namestitev nedelujoče verzije mikrostoritve, s čimer se izognemo izpadu delovanja aplikacije, posledično pa izboljšamo proces razvoja programske opreme.

Ključne besede: mikrostoritve, dinamično posodabljanje, odpornost na napake, decentralizacija.

Abstract

Applications that require a high availability are updated with dynamic software updating methods, which do not result in any downtime during the update process. Dynamic updates in the microservice architecture are typically coordinated by a centralised container orchestrator. This paper describes the results of the work in progress in which we are developing a decentralised method for dynamic updates of microservices. The proposed method defines update coordinators for the decentralised coordination of the updates process. Update coordinators use fault-tolerance mechanisms to evaluate the newly deployed microservice version. The evaluation prevents the deployment of a faulty microservice version and consequent service outage, all of which improve the software development process.

Keywords: Microservices, dynamic software updating, fault tolerance, decentralisation.

1 UVOD

Sodobne aplikacije morajo zagotavljati visoko stopnjo razpoložljivosti, prav tako pa jih je potrebno stalno posodabljati, na primer zaradi nadgrajevanja funkcionalnosti ali odprave napak. Z uporabo metod dinamičnega posodabljanja lahko dosežemo ničelni čas izpada storitev aplikacije med njenim posodabljanjem, vendar je metode potrebno prilagoditi sistemu in arhitekturi aplikacije. V arhitekturi mikrostoritev za dinamično posodabljanje običajno uporabljamo orodja za orkestracijo vsebnikov, ki so centralizirana in med procesom posodabljanja ne zagotavljajo naprednih mehanizmov za odpornost na napake. V članku opišemo povzetke izvirnega raziskovalnega dela, ki je še v teku, v katerem razvijamo metodo za dinamično posodabljanje mikrostoritev, s katero želimo izboljšati pomanjkljivosti obstoječih pristopov. Cilj raziskovalnega dela je razviti metodo, ki bo postopek dinamičnega posodabljanja koordinirala decentralizirano brez centralnega orodja za orkestracijo in pri tem evalvirala novo verzijo mikrostoritve, s čimer bo preprečila names-

titev nedelujoče verzije in posledično izpad delovanja celotne aplikacije.

2 SODOBNI PRISTOPI K RAZVOJU PROGRAMSKE OPREME

Popularizacija računalništva v oblaku in selitev programske opreme v oblak sta vplivali na način razvoja in arhitekturo programske opreme. S ciljem boljše izkoriščenosti virov in uporabe podpornih oblačnih mehanizmov, kot so samodejno skaliranje, preverjanje vitalnosti in odpornost na napake, so razvili nove arhitekturne pristope, optimizirane za izvajanje v oblaku (Kratzke & Quint, 2017). Posledično so se razvili tudi novi pristopi k razvoju, testiranju in nameščanju programske opreme.

Najbolj razširjen pristop k razvoju programske opreme za izvajanje v oblaku je arhitektura mikrostoritev, v kateri je posamezna aplikacija sestavljena iz večjega števila neodvisnih storitev – mikrostoritev. Razbitje aplikacije na mikrostoritve sledi poslovnim domenam, pri čemer je posamezna mikrostoritev odgovorna za implementacijo ene funkcionalnosti. Posamezna mikrostoritev je pakirana kot vsebnik in nameščena v večjem številu instanc, kar zagotavlja visoko odzivnost in razpoložljivost. Mikrostoritve med sabo komunicirajo preko jasno definiranih vmesnikov na standardnih omrežnih protokolih. Omrežni naslovi instanc mikrostoritev so shranjeni v registru storitev, iz katerega se pridobivajo za potrebe komunikacije med mikrostoritvami. Mikrostoritvene aplikacije so torej distribuirane, njihova modularna zasnova pa omogoča učinkovito skaliranje, pri čemer je posamezne dele aplikacije možno skalirati glede na njihovo obremenjenost (Esposito, Castiglione & Choo, 2016).

Vsaka mikrostoritev ima jasno določeno ekipo, ki je odgovorna za celoten življenjski cikel mikrostoritve, od razvoja in testiranja do nameščanja in vzdrževanja mikrostoritve v vseh okoljih (Zhu, Bass & Champlin-Scharff, 2016). Zaradi neodvisnosti med posameznimi mikrostoritvami, majhnih razvojnih ekip in odgovornosti ekip za celoten življenjski cikel mikrostoritve razvijalci pogosto uporabljajo kratke razvojne cikle s pogostimi namestitvami novih verzij v produkcijsko okolje. Razvojni cikli so podprti z orodji za avtomatizacijo, ki zagotavljajo zvezno integracijo in nameščanje, pri čemer se za nameščanje novih verzij uporablja koncept dinamičnega posodabljanja, ki med prehodom na novo verzijo zagotavlja neprekinjeno delovanje mikrostoritve (O'Connor, Elger & Clarke, 2017).

Razbitje aplikacije na mikrostoritve, njihov neodvisen razvoj in uporaba kratkih razvojnih ciklov z manj celovitega testiranja povečajo potencial za vnos nepredvidenih napak. Slednje se lahko pojavijo zaradi nepravilnih sprememb vmesnikov ali izpada delovanja mikrostoritve zaradi nepravilne konfiguracije izvajalnega okolja. Posledica so napake pri komunikaciji med mikrostoritvami, ki morajo delovati kot celota in zagotavljati kompozitne funkcionalnosti, sestavljene iz funkcionalnosti posameznih mikrostoritev. Mikrostoritve morajo posledično imeti visoko stopnjo odpornosti na napake, ki nemoteno delovanje aplikacije zagotavlja tudi v primeru pojava nepredvidenih napak. Na primer, neuspešen klic mikrostoritve ne sme povzročiti izpada delovanja celotne aplikacije (Killalea, 2016). Mehanizmi za povečanje odpornosti na napake vključujejo preverjanje vitalnosti mikrostoritev, samodejne ponovne klice zunanjih odvisnosti, vpeljavo maksimalnega odzivnega časa, prekinjevalce toka, pregrade in določanje alternativnih operacij, ki se izvedejo v primeru napak. Dotični mehanizmi so tako eden izmed ključnih gradnikov arhitekture mikrostoritev, saj predstavljajo protiutež povečanemu potencialu za napake, ki ga vpelje razbitje aplikacije na več neodvisnih izvajalnih enot (Toffetti, Brunner, Blöchlinger, Spillner & Bohnert, 2017).

3 IZZIVI DINAMIČNEGA POSODABLJANJA MIKROSTORITEV

Dinamično posodabljanje programske opreme je proces, v katerem namestimo novo verzijo aplikacije, ne da bi s tem prekinili zagotavljanje njenih storitev (Hicks & Nettles, 2005). V arhitekturi mikrostoritev posodabljamo posamezne mikrostoritve, in sicer tako, da obstoječo verzijo mikrostoritve nadomestimo z novo verzijo.

Obstoječi načini dinamičnega posodabljanja mikrostoritev temeljijo na orodjih za orkestracijo vsebnikov. Ta tipično upravljajo elastičnost aplikacije in zagotavljajo kakovost storitev, lahko pa so zadolžena tudi za dinamično posodabljanje. V našem delu nas zanima izključno vloga orodij za orkestracijo kot upravljalcev dinamičnega posodabljanja. V procesu dinamičnega posodabljanja skrbnik naloži novo verzijo mikrostoritve, orodje za orkestracijo pa zažene vsebnik z novo verzijo, preveri njegovo delovanje, nanj preusmeri omrežni promet ter odstrani vsebnike s prejšnjo verzijo. Orodja za orkestracijo vsebni-

kov predstavljajo centralizirano rešitev za dinamično posodabljanje, pri čemer je nemoteno delovanje sistema odvisno od nemotenega delovanja orodja za orkestracijo. Prvi cilj našega raziskovalnega dela se osredotoča na razvoj metode za decentralizirano koordinacijo dinamičnega posodabljanja mikrostoritev, s katero želimo nadomestiti centralizirane posodobitvene metode orodij za orkestracijo.

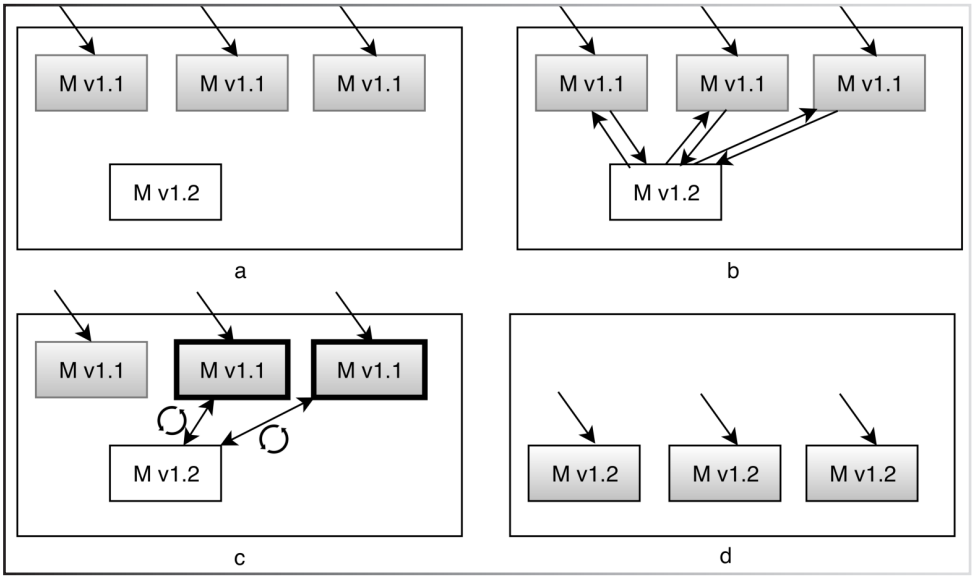
Visoke zahteve po odpornosti na napake v arhitekturi mikrostoritev veljajo tudi za proces dinamičnega posodabljanja, za katerega želimo, da je odporen na napake, ki se lahko pojavijo kot posledica namestitve neustrezne ali nekompatibilne verzije mikrostoritve. Z ustreznimi mehanizmi za povečanje odpornosti na napake želimo preprečiti izpad delovanja aplikacije med posodobitvijo. S tem posledično izboljšamo proces razvoja programske opreme, saj razvijalcem omogočimo, da uporabljajo kratke razvojne cikle s pogostimi namestitvami brez tveganja za izpad delovanja aplikacije zaradi napake v novi verziji. Drugi cilj našega raziskovalnega dela je razviti mehanizme za povečanje odpornosti na napake pri uporabi decentralizirane metode za dinamično posodabljanje mikrostoritev.

4 PREDLOG METODE ZA DECENTRALIZIRANO IN ODPORNO DINAMIČNO POSODABLJANJE MIKROSTORITEV

Za izpolnitev zastavljenih raziskovalnih ciljev predlagamo metodo za decentralizirano in odporno dinamično posodabljanje mikrostoritev. Predlagana metoda je namenjena posodobitvam z manjšimi spremembami mikrostoritev, ki so pogoste v sodobnih razvojnih ciklih. Sem spadajo odprava napak, manjše spremembe ali dopolnitve poslovne logike in dopolnitve vmesnikov, ki ne porušijo združljivosti s prejšnjimi verzijami.

Za decentralizacijo dinamičnega posodabljanja predlagamo vpeljavo komponente za koordinacijo, ki postane sestavni del vsake mikrostoritve. S tem odgovornost za dinamično posodabljanje iz centraliziranih orodij za orkestracijo vsebnikov prestavimo na mikrostoritve. Namesto potrebe po centralnem upravljanju dinamičnega posodabljanja se obstoječa in nova verzija mikrostoritve sami dogovorita o postopku posodobitve, pri čemer posodobitev koordinirajo instance obstoječe verzije, ki jih imenujemo koordinatorji posodobitve. Ker je obstoječa verzija mikrostoritve nameščena v večjem številu instanc, je zagotovljena visoka odpornost na napake, saj se postopek dinamičnega posodabljanja uspešno zaključi tudi ob izpadu enega ali več koordinatorjev posodobitve.

Za povečanje odpornosti na napake v procesu dinamičnega posodabljanja definiramo fazo evalvacije nove verzije. V fazi evalvacije so zahtevki odjemalcev mikrostoritve še naprej usmerjani na instance obstoječe verzije, nova verzija pa je aktivno evalvirana s strani koordinatorjev posodobitve. Ti s klici API-ja za preverjanje vitalnosti testirajo odzivnost nove verzije



Slika 1: **Shema predlagane metode za decentralizirano dinamično posodabljanje z evalvacijo nove verzije mikrostoritve.**

in preverijo rezultate kontrol vitalnosti, ki predstavljajo teste funkcionalnosti mikrostoritve. V primeru pozitivne evalvacije je nova verzija sprejeta, v primeru negativne evalvacije pa zavrnjena brez negativnega vpliva na delovanje celotne aplikacije.

Shema na sliki Slika 1 povzema štiri glavne korake predlagane metode. Pred posodobitvijo so zahtevki odjemalcev mikrostoritve *M* posredovani instancam verzije *v1.1*, saj so te vpisane v registru storitev. Cilj posodobitve je zamenjati instance mikrostoritve *M* verzije *v1.1* z instancami verzije *v1.2*. Instance verzije *v1.1* implementirajo predlagano komponento za koordinacijo, zato lahko prevzamejo vlogo koordinatorjev posodobitve – obstoj instanc s komponento za koordinacijo je predpogoj za začetek procesa dinamičnega posodabljanja s predlagano metodo. V prvem koraku (a) razvijalec v izvajalno okolje namesti vsebnik z novo verzijo mikrostoritve *M* verzije *v1.2*. Instanca verzije *v1.2* ni vpisana v register storitev, zato ne dobiva zahtevkov odjemalcev. V drugem koraku (b) verzija *v1.2* iz registra storitev pridobi naslove instanc mikrostoritve *M* verzije *v1.1* in za koordinatorje posodobitve izbere najbolj odzivne instance. V tretjem koraku (c) izbrani koordinatorji posodobitve s pomočjo API-ja za preverjanje vitalnosti evalvirajo verzijo *v1.2*. Za uspešno evalvacijo mora verzija *v1.2* v določenem času odgovoriti s pozitivnimi statusi posameznih kontrol vitalnosti. Glede na rezultate evalvacije je nova verzija *v1.2* bodisi zavrnjena bodisi sprejeta. Četrti korak (d) prikazuje pozitivno evalvacijo in sprejetje verzije *v1.2*. V primeru pozitivne evalvacije se instance verzije *v1.2* registrirajo v register storitev in posledično začnejo prejemati zahtevke odjemalcev. Instance verzije *v1.1* se iz registra odstranijo in prekinejo svoje izvajanje, njeni vsebniki pa so posledično samodejno ustavljeni. V primeru negativne evalvacije je nova verzija zavrnjena, njene instance prekinejo svoje izvajanje, njeni vsebniki pa so posledično ustavljeni.

Predlagana metoda ne zahteva sprememb obstoječe infrastrukture, mikrostoritve pa ni potrebno izvajati v priviligiranem načinu, saj že imajo vse potrebne pravice za izvajanje metode – pisanje v register storitev, prekinitev lastnega izvajanja in komunikacija z ostalimi instancami iste mikrostoritve, ki je ustrezno zaščitena z mehanizmi avtentikacije. V predlaganem procesu dinamičnega posodabljanja vsaka mikrostoritev tako manipulira le z lastnimi instancami, nasprotno pa imajo orodja za orkestracijo pregled in privilegij upravljanja vseh mikrostoritev v izvajalnem okolju.

Predlagana metoda je definirana v obliki specifikacije, ki jo lahko implementira poljubna mikrostoritev, pri čemer je metoda tipično implementirana kot razširitev mikrostoritvenega ogrodja. Z uporabo razširjenega ogrodja lahko nato razvijalci brez dodatnega dela razvijajo mikrostoritve, ki že vsebujejo implementacijo komponente za koordinacijo in so same sposobne izvajati decentralizirano dinamično posodabljanje z evalvacijo novih verzij. Z razširitvijo obstoječih mikrostoritvenih ogrodij razvijalcem omogočimo enostavno uporabo razvite metode brez potrebe po dodatnem programiranju ali uporabi namenskih centraliziranih orodij za orkestracijo vsebnikov.

5 NAČRT EVALVACIJE IN PREDVIDENI REZULTATI

Predlagano metodo za decentralizirano in odporno dinamično posodabljanje mikrostoritev bomo evalvirali s prototipno implementacijo v obliki razširitve izbranega odprtokodnega ogrodja za razvoj mikrostoritev. Razširjeno ogrodje bomo nato uporabili za izdelavo testne aplikacije, ki jo bomo evalvirali s simulacijami procesa dinamičnega posodabljanja.

V preliminarni evalvaciji smo pripravili prvo prototipno implementacijo predlagane metode ter jo evalvirali s simulacijami dinamičnega posodabljanja preproste mikrostoritvene aplikacije. Pokazali smo, da predlagana metoda uspešno izvede proces dinamičnega posodabljanja brez centraliziranega orodja za orkestracijo. V nadaljnji evalvaciji bomo izboljšali prototipno implementacijo predlagane metode ter definirali več kompleksnejših simulacij. Simulirali bomo nedelujoče koordinatorje posodobitve, s čimer želimo pokazati, da metoda deluje tudi v primeru enega ali več nedelujočih koordinatorjev. Dokazati želimo, da predlagana metoda izboljša robustnost dinamičnega posodabljanja mikrostoritev, zato bomo simulirali namestitve neustreznih verzij mikrostoritev in merili pravilnost evalvacije ter sprejetja ali zavrnitve nove verzije. Želimo se prepričati še, da predlagana metoda nima negativnih vplivov na učinkovitost izvajanja dinamičnih posodobitev, zato bomo merili čas, potreben za vzpostavitev koordinatorjev posodobitve, čas, potreben za izvedbo evalvacije nove verzije, in celoten čas izvedbe posodobitvenega procesa. Pričakujemo, da se proces posodobitve zaključi v enakem ali krajšem času kot pri uporabi centraliziranih orodij za orkestracijo, torej v rangu

nekaj sekund, pri tem pa znatno izboljša odpornost na napake, saj proces posodobitve deluje tudi v primeru izpada koordinatorjev posodobitve ali namestitve neustrezne verzije mikrostoritve.

6 SKLEP

V članku smo povzeli izsledke raziskovalnega dela, v katerem razvijamo metodo za dinamično posodabljanje mikrostoritev, ki deluje brez potrebe po centraliziranem orodju za orkestracijo vsebnikov in izboljšuje odpornost na napake v fazi dinamičnega posodabljanja. Predlagana metoda razbremenjuje razvijalce programske opreme, saj jim omogoča uporabo kratkih razvojnih ciklov s pogostimi namestitvami v produkcijsko okolje, pri čemer morebitna namestitev nekompatibilne ali nedelujoče verzije mikrostoritve ne more povzročiti izpada delovanja celotne aplikacije.

V nadaljevanju raziskave nameravamo definirati vse podrobnosti predlagane metode ter jo evalvirati s pomočjo prototipne implementacije in simulacij. Pričakujemo, da bo razvita metoda pozitivno vplivala na odpornost na napake v procesu dinamičnega posodabljanja mikrostoritev.

LITERATURA

[1] Esposito, C., Castiglione, A., & Choo, K. K. R. (2016). *Challenges in delivering software in the cloud as microservices.* IEEE Cloud Computing, 3(5), 10–14. https://doi.org/10.1109/MCC.2016.105

[2] Hicks, M., & Nettles, S. (2005). *Dynamic software updating. ACM Transactions on Programming Languages and Systems* (TOPLAS), 27(6), 1049-1096. https://doi.org/10.1145/1108970.1108971

[3] Killalea, T. (2016). *The hidden dividends of microservices.* Communications of the ACM, 59(8), 42–45. https://doi.org/10.1145/2948985

[4] Kratzke, N., & Quint, P. C. (2017). *Understanding cloud-native applications after 10 years of cloud computing-a systematic mapping study.* Journal of Systems and Software, 126, 1–16. https://doi.org/10.1016/j.jss.2017.01.001

[5] O'Connor, R. V., Elger, P., & Clarke, P. M. (2017). *Continuous software engineering—A microservices architecture perspective.* Journal of Software: Evolution and Process, 29(11), e1866. https://doi.org/10.1002/smr.1866

[6] Toffetti, G., Brunner, S., Blöchlinger, M., Spillner, J., & Bohnert, T. M. (2017). *Self-managing cloud-native applications: Design, implementation, and experience.* Future Generation Computer Systems, 72, 165–179. https://doi.org/10.1016/j.future.2016.09.002

[7] Zhu, L., Bass, L., & Champlin-Scharff, G. (2016). *DevOps and its practices.* IEEE Software, 33(3), 32–34. https://doi.org/10.1109/MS.2016.81

▪

Jan Meznarič je asistent in raziskovalec na UL FRI. Raziskovalno se ukvarja z mikrostoritvami in ostalimi koncepti cloud-native arhitekture, s poudarkom na zvezni integraciji, odpornosti na napake, centralni konfiguraciji, odkrivanju storitev in zbiranju metrik. Raziskovalno in aplikativno sodeluje pri večjem številu projektov in je aktivno udeležen pri razvoju odprtokodnega ogrodja za razvoj mikrostoritev KumuluzEE.

▪

Matjaž B. Jurič je predstojnik Laboratorija za integracijo informacijskih sistemov na UL FRI in mentor start-up podjetij. Je avtor 17 knjig, izdanih pri mednarodnih založbah ter več kot 600 drugih publikacij. Vodil je številne raziskovalne in aplikativne projekte, ponaša pa se tudi s prestižnimi nazivi Java Champion, IBM Champion in Oracle ACE Director. Prejel je več mednarodnih nagrad, med drugim nagrado za najboljšo SOA knjigo (New York), nagrado za najboljši SOA projekt v telekomunikacijah (Las Vegas), nagrado Java Duke's Choice Award Winner (San Francisco) za najboljšo inovacijo v Javi, nagrado za najboljši znanstveni članek s področja storitev, nagrado za najboljšega raziskovalca po mnenju industrije in Zlato plaketo za izjemne zasluge pri razvijanju znanstvenega ustvarjanja.

Resource saving technologies in education: A step towards a green society

Inna Pavlova[1], Alenka Kavčič[2]
[1]North-Caucasus Federal University, 355017, Stavropol, Pushkin street 1
[2]University of Ljubljana, UL FRI, Večna pot 113, 1000 Ljubljana
shinnessa@mail.ru, alenka.kavcic@fri.uni-lj.si

Izvleček

Članek obravnava aktualne probleme moderne družbe – degradacijo okolja in neučinkovito rabo virov, kot tudi možne načine ohranjanja virov – z vidika izobraževanja. V prvem delu na kratko predstavimo pregled literature s področja zelenih tehnologij v izobraževanju. Sledi opis didaktičnega modula za usposabljanje bodočih učiteljev informatike v aktivnostih varčevanja z viri. Namen modula je spodbujanje njihove pripravljenosti za uporabo novih izobraževalnih metod v šolah, s posebnim poudarkom na zelenih informacijskih tehnologijah, kot tudi razvoj splošne kulture varčevanja z viri.

Ključne besede: IKT v izobraževanju, usposabljanje bodočih učiteljev, varčevanje z viri, zelena informacijska tehnologija.

Abstract

This article discusses the actual problems of modern society – environmental degradation and inefficient use of resources, as well as possible ways of resource conservation – from the educational perspective. A brief literature review of green technologies in education is presented in the first part. Next, we describe a didactic module for training future informatics teachers in resource-saving activities. The module aims to form their readiness to use novel educational methods in schools with a special focus on green information technologies as well as to develop a general resource-saving culture.

Keywords: Green computing, ICT in education, resource saving, training future teachers.

1 INTRODUCTION

Considering the existing environmental degradation, improving environmental performance, combating global warming and improving the efficiency of resource management are priorities on the list of global problems that need to be addressed very quickly. One of the main tasks of the governments around the world is to control and reduce the negative impact of harmful substances on the environment, and balance consumption of natural resources. However, because of rapid technological progress, just the opposite is happening: more people are using more devices, increasing the energy consumption, and more resources are required to perform any technical action, as they have become more complex. Many governmental programs and initiatives have been prepared all around the world with the main objective of more efficient use of energy and consequently reducing its total costs, which should also have a positive impact on the world economy and society as a whole.

Information and communication technologies (ICT) that are already interweaved into all our everyday activities also represent a major source of environmental pollution. Thus, the new trends are moving towards using eco-technologies or green computing, with the main purpose of creating environmentally friendly and sustainable computer technology. Interest in green information technologies has recently increased in all countries, including Europe and Russia (Dastbaz, Pattinson, & Akhgar, 2015).

2 LITERATURE ON GREEN IT IN SCHOOLS

Green information technology (green IT) as a practice of environmentally sustainable computing is becoming an important issue and the problems associated with it raise growing concerns about the negative impact of digital society on the environment. The problem of environmentally sustainable computing and resource saving is still poorly studied, and there is a limited number of sources in the literature, conferences and seminars on this topic.

Green or sustainable computing helps reduce energy consumption and losses to the environment when using computer equipment, extends the service life of the product, and makes it more energy efficient, while the production waste is easily recycled or biodegradable, with a lower content of hazardous substances. Thus, the sustainable development of information technology (IT) plays an important role in protecting our future. Recently, more attention to the "greening" of information technology and information systems (IS) has become apparent worldwide.

3 COURSES TACKLING THE SUSTAINABILITY ISSUES

In response to growing environmental concerns, more and more organizations are adopting environmental initiatives. In particular, many higher education institutions (HEIs) have implemented various security initiatives on campus through policies as well as teaching and learning. Master's and postgraduate programs provide training in a wide range of information technologies along with sustainable strategies to teach students how to create and maintain systems while reducing their harm to the environment. Universities such as Australian National University, Athabasca University in Canada, and Leeds Beckett University in the UK offer engineering programs *Sustainable ICT development*, *Green ICT strategies*, *MSc Sustainable Computing* in full and partial access (Dastbaz, Pattinson, & Akhgar, 2015).

In Sweden, the University of Chalmers has created a special course (Chalmers, 2017), which aims to inspire and encourage students to think about how they can contribute to sustainable development, in everyday and professional life. A basic understanding of the concept of sustainable development is needed, and knowledge of how the current use of natural resources and ecosystem services by humans is unsustainable, as well as possible strategies and solutions to enhance sustainability. Thus, this course should arouse great interest in sustainability issues, and provide the student with the knowledge and tools necessary to address the complex sustainability issues in their future professional lives.

4 GREEN EDUCATION

The review of the literature on the green IT research reveals the fact that ensuring the implementation of the necessary requirements should begin with the formation of human needs for the rational use of resources at the initial stage of education. That is, school education should enable the students to develop the type of thinking, which will be aimed at resource saving and respect for nature, while based on modern teaching methods. The authors of the book *Green Teaching and Learning in Schools* (Blendinger, Hailey, & Shea, 2015) justify the importance of creating a culture of teaching and learning in the school to achieve a strong impact on students in relation to the preservation of the environment. They present successful and practical examples, "teaching green" in action, which have been provided to practicing teachers in primary and secondary schools.

In their study on Green Education, Aithal and Rao (2016) show that educational institutions are also working to achieve sustainability. They discuss that growth or learning should only take place in an environment that promotes development in children's lives. Teachers and students need to be aware of and apply environmentally sound practices in their learning, and a culture of preservation should be an integral part of the curriculum.

In the article on green IT at HEI, researchers Suryawanshi and Narkhede (2014) argue that green information and communication technologies are an innovative approach to the use of ICT related to environmental protection and ICT sustainability in the future, and represent a practice of achieving corporate social responsibility by minimizing carbon footprint, ICT waste and energy savings. The rational use of green ICT in education is also analysed, and critical success factors of their implementation are identified based on a survey of individual educational institutions and interviews with the key academic experts in India. The study identify seven critical success factors that are important for the sustainability of ICT in the future: optimal use of resources, stakeholder participation, renewable energy, energy

conservation, institutional policies, green ICT Committee activities, and legislation.

Against the background of the global development of green technologies, also China made the environmental protection a key state policy and included it in the strategy of national sustainable development (Ximing & Chunzhao, 2011). The study of Ximing and Chunzhao (2011) shows that students at universities contribute to environmental awareness, but they are not motivated to participate in environmental programs and activities. They conclude that HEIs need to combine theory with practice to make the process of learning more interesting. To improve the quality of environmental education, universities should also update and improve the philosophy of teaching, personnel training plan, contents and methods of teaching, and teachers training. More needs to be done to explore effective methods and ways of introducing green education and highly skilled staff in the new environment.

Another researcher Lu Chen (2017) discusses in his paper on resource saving higher education development that the thorough exploration of the concept of resource saving and a comprehensive study of its internal meaning and intrinsic value are the key to the optimization and integration of higher education resources. In order to save resources, it is possible to effectively optimize the concept of industrialization and integration of higher education resources, and promote the integration and optimization of regional industrialization resources, thereby forming a significant impact on the diversity and multi-channel of the development of higher education resources.

With the aim of better understanding the research area of green IT, Asadi, Hussin, & Dahlan (2017) conducted a literature review from 2007 to 2016 on the green IT research. They note that current research covers numerous topics within green IT, especially initiation, approaches and strategies, adoption frameworks and benefits, but other topics receive insufficient attention. By asking research questions, they help scientists identify rigorous research areas for further study. The results of their research provide a roadmap to guide future research on green IT and highlight areas for successful implementation of green IT.

Researchers Singh and Sidhu (2016) conclude in their review article that green computing seeks to reduce the unwanted and harmful effects of computers on the environment by reducing air, water and soil pollution. With ever-increasing research in science and technology, it is possible to overcome obstacles. They also argue that each of us has to take small steps towards taking green computing measures in order to enable a healthy growth of our environment.

5 RESOURCE SAVING AND EDUCATIONAL MODELS

Information and communication technologies incorporated into the modern teaching methods have significantly changed the educational system, but have not reduced the value of traditional learning. The best part of digital education in the 21st century is that it combines two aspects: traditional classroom learning and learning with innovative technologies. Both approaches are complementary and support each other. The use of ICT in education is also a way to save resources. For example, a platform for online training restricts frivolous use of paper, directly reducing the felling of trees, while a free software might save material resources of users. Cloud storage and intelligent search engines make it easier for students to access a wider range of information sources, without time (and money) consuming browsing through a large number of books to find specific information.

However, data centres are vast consumers of electricity and therefore under an increasing pressure to reduce carbon dioxide emissions. In response, cloud service providers have begun to set sustainable development goals by using renewable sources in their services (Mann, 2016).

6 RESOURCE-SAVING EDUCATION

We can argue that the resource saving is a modern stage of evolution in the context of transformation of the education model. Based on the universal principles, it is currently possible to form a comprehensive resource-saving education. The innovative task of resource saving is a professionally- and problem-oriented educational task of saving and optimizing costs and reserves. It contributes to the formation of relevant components of the resource saving competencies. Its result also determines a set of measures for introducing new technologies to improve the efficiency of education.

Innovative tasks of resource saving can be a means of literacy formation of future graduates. The professional education, based on the competences, should

approach the complexity of resource saving tasks by focusing on updating the resource values and rising awareness of the need for reducing their consumption.

An efficient solution to these problems can be obtained only through information and methodological readiness of teachers for the use of new resource-saving forms of education (e.g. distance, electronic, mobile, co-education), the creation of electronic educational resources, and providing access to these resources to a wide range of students, consequently improving the availability and quality of education (Konopko & Pankratova, 2017).

7 A MODULE FOR TRAINING TEACHERS ON RESOURCE SAVING TECHNOLOGIES

The North Caucasus Federal University (NCFU) in Russia introduced key concepts and ideas of green IT in the educational process of undergraduate students majoring in Computer Science. A distance-learning module, available on a learning management system of NCFU training (https://el.ncfu.ru), allows lecturers to apply the practice of e-learning and blended learning in the classroom. The educational module titled *European resource saving trends in Computer Science teacher training* was developed within the Erasmus+ Jean Monnet project.

The module implements an individual approach in the formation of various components of readiness for resource-saving activities of students (i.e. future teachers of computer science). At the theoretical level, axiological and cognitive components of readiness are formed via video lectures, webinars and on-line discussions, while the activity and management components of readiness are formed at the practical level. Axiological components of readiness relate to the system of values of the individual, motivating the student for resource-saving activities, consequently forming a positive attitude to it and grow the desire to spread the experience of resource saving. Cognitive components of readiness represent the necessary competencies on the theoretical and methodological foundations of resource saving. Activity components of readiness relate to formation of skills and resource-saving activities, while management components of readiness refer to the development of student skills to predict, plan, organize, control, analyse and reflect resource-saving activities.

Practical work (in the form of project activities, debates, discussions or round tables, organized with the use of remote technologies) contributes to the acquisition of students' skills of resource-saving activities. In addition, the methodological novelty of the project is the partnership of lecturers and students in the form of project activities, forums, online discussions, webinars, blogs, scientific consultations and joint publications. The virtual platform of the module allows students to have unlimited online access to all educational and methodical materials of the module. Besides, it provides an opportunity to share the results of applied research and experimental educational work to students and lecturers.

The actual effects of the module can be assessed only after the practical application of acquired knowledge and competences, with the new generation of teachers of Informatics in secondary schools (Konopko & Pankratova, 2017).

8 CONCLUSION

Applying green IT leads to lower energy consumption, carbon emissions and environmental impacts, as well as lower costs for organizations on the long run. The ultimate goal of going green is sustainability and addressing adverse environmental impacts. However, using green technology should be the motivation from the point of view of social responsibility. By combining traditional learning with green IT, it is possible to use the best of what green and human-driven education has to offer, creating a learning experience that keeps pace with technology while minimizing the consumption of various resources. It is important to remember the role that the human teacher will always play in the classroom. After all, teachers have unique and personal information about each student's progress, act as role models and local experts, and provide inspiration in ways that technology cannot.

REFERENCES

[1] Aithal, P. S., & Rao, P. (2016). Green Education Concepts & Strategies in Higher Education Model. *International Journal of Scientific Research and Modern Education (IJSRME) ISSN (Online)*: 2455 – 563, 1(1), 793-802.

[2] Asadi, S., Hussin, A. R. C., & Dahlan, H. M. (2017). *Organizational research in the field of Green IT: A systematic literature review from 2007 to 2016*. Telematics and Informatics, 34(7), 1191-1249.

[3] Blendinger, J., Hailey, L. A., & Shea, D. (2015). *"Green" Teaching and Learning in Schools. Marketing the Green School: Form, Function, and the Future*, 183-193.

[4] Chalmers (2017). *Our role in sustainable development.* Retrieved from https://www.chalmers.se/en/about-chalmers/

Chalmers-for-a-sustainable-future/Pages/Sustainability-at-Chalmers.aspx
[5] Chen, L. (2017). *Idea of Resource Saving Higher Education Development on the Integration and Optimization of Educational Resources.* Eurasia Journal of Mathematics, Science and Technology Education, 13(10), 7071-7076.
[6] Dastbaz, M., Pattinson, C., & Akhgar B. (2015). *Green information technology. A sustainable approach.* Amsterdam: Elsevier Morgan Kaufmann.
[7] Konopko, E. A., & Pankratova, O. P. (2017). *Modeling the process of preparing future informatics teachers for resource-saving activities based on European experience.* International Journal of Modern Information Technologies and IT Education, 13(4), 134140.
[8] Mann, S. (2016). *A Rethink for Computing Education for Sustainability.* Paper presented at the International Conferences on Internet Technologies & Society (ITS), Education Technologies (ICEduTECH), and Sustainability, Technology and Education (STE), pp. 221-228.
[9] Singh, M., & Sidhu, A. S. (2016). *Green Computing.* International Journal of Advanced Research in Computer Science, 7(6), 195–197.
[10] Suryawanshi, K., & Narkhede, S. (2014). *Green ICT at Higher Education Institution: Solution for Sustenance of ICT in Future.* International Journal of Computer Applications, 107(14), 35-38.
[11] Ximing, S., & Chunzhao, L. (2011). *Survey of environmental education (EE): case study of higher education institutions in Ningbo.* Energy Procedia, 5, 1820–1826.

▪

Inna Pavlova is a second year master's student at the Department of Informatics, North Caucasus Federal University, Russia. Her research interests are in the field of information and communication technologies in education, cloud technologies, virtual reality, and resource-saving technologies.

▪

Alenka Kavčič, Ph.D. is a senior lecturer at the Faculty of Computer and Information Science, University of Ljubljana. Beside her pedagogical responsibilities, she is involved in a number of research projects related to multimedia and internet technologies, human-computer interaction, and computer-based education and learning, especially the innovative use of information technologies in education. She has co-organized several Summer Schools for high school teachers and students.

Samodejno preverjanje pravilnosti študentskih nalog iz programiranja

Aleš Čep, Damijan Novak, Jani Dugonik
Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Koroška cesta 46, 2000 Maribor
ales.cep@um.si, damijan.novak@um.si, jani.dugonik@um.si

Izvleček

Fakultete se lahko soočijo s problematiko preverjanja pravilnosti delovanja velikega števila oddanih nalog, predvsem pri predmetih iz programiranja. Predlagan sistem za avtomatizacijo preverjanja oddanih nalog lahko čas testiranja nalog bistveno skrajša in s tem razbremeni pedagoško osebje. Povratne informacije o točnosti oddane naloge so deležni tudi študenti, ki jim sistem poda opis napake, ki jo lahko v ponovnem poskusu oddaje odpravijo. Sistem omogoča preverjanje nalog v dveh programskih jezikih (C# in C++), komunikacija med pedagogom in študentom pa spada v kategorijo e-učenja. Prednost uporabe predlaganega sistema je popolna nepristranost ocenjevalca, slabost pa se kaže v večjem začetnem časovnem vložku, potrebnem za sestavo naloge. V članku bo podrobneje predstavljena notranja zgradba sistema za preverjanje pravilnosti delovanja nalog iz programiranja.

Ključne besede: Docker, e-učenje, Moss, ocenjevanje študentskih nalog, testiranje enot, testno voden razvoj.

Abstract

Faculties may face the problem of verification of many submitted assignments, especially in subjects with programming assignments. The proposed system for verification automation of submitted assignments can significantly reduce the testing times and relieve the teaching staff. Feedback about the submitted assignments' correctness is also given to the students, as well as descriptions of the errors so that they can correct the assignments and submit them again. The system supports the verification for two different programming languages (C# and C++). The communication between the teacher and the student puts the proposed system into the E-learning category. The advantage of using the proposed system is the complete impartiality of the assessor while the disadvantage is that more time is needed to compose an assignment. This paper presents the detailed internal structure of the assignment verification system.

Keywords: Docker, e-learning, grading of the student assignments, Moss, Test Driven Development, Unit testing.

1 UVOD

Pri predmetih iz programiranja se študente seznani s pravilno sintakso in uporabo specifičnih programskih jezikov, s pristopi sestavljanja postopkov oz. algoritmov ter s predstavitvijo podatkov in podatkovnih struktur.

Na fakultetah pri predmetih z večjim številom študentov (več kot 100) predstavlja preverjanje oddanih nalog velik izziv tako z vidika zagotavljanja enakega kriterija ocenjevanja kot tudi časovne obremenitve pedagoškega osebja. Zaradi teh razlogov smo se odločili za avtomatizacijo postopka preverjanja študentskih nalog, ki se že nekaj let uporablja na Fakulteti za elektrotehniko, računalništvo in informatiko. Sistem se uporablja pri predmetu Podatkovne strukture v 1. letniku visokošolskega študija, na dveh študijskih smereh (računalništvo in informatika), kjer je v posameznem študijskem letu skupno vpisanih več kot 200 študentov. Študentje lahko izbirajo med dvema programskima jezikoma: C# in C++. Pri vsaki nalogi so na voljo navodila in predloga programa, ki vsebuje vnaprej deklarirane razrede, programske strukture ter metode. Metode, ki jih morajo implementirati študenti na osnovi podanih navodil, so podane samo s prototipi in komentarji z razlago.

Od samega začetka je bil sistem zasnovan z mislijo na vpetost študentov v učni proces in na njihovo uporabniško izkušnjo. Če študent pri implement-

aciji sledi navodilom naloge, mu sistem ob morebitni storjeni napaki pri osvajanju zahtevane snovi oziroma opravljanju naloge poda jasen povraten odziv. Pedagog in študent sta tako aktivno vključena v prepletenost učnega procesa in Informacijsko-Komunikacijske Tehnologije (IKT) sistema, kar na kratko imenujemo e-učenje.

Predlagan sistem nadaljuje razvoj sistema za preverjanje pravilnosti študentskih nalog (Zemljak, Novak, Čep, & Verber, 2016). Glavni napredek je storjen v stopnji avtomatizacije s prenosom nalog s spletne učilnice in pri sestavljanju razpredelnice z odzivi, kar pohitri objavo rezultatov. Prav tako smo v sistem dodali zabojnike Docker, s čimer smo poskrbeli za dodatno izolacijo med izvajanjem testov. Za preverjanje podobnosti izvorne kode smo dodali uporabo sistema Moss.

Preostanek članka je organiziran na naslednji način. Poglavje **2** opiše področja in orodja, uporabljena v našem sistemu. Poglavje **3** predstavi sorodne sisteme s področja računalniškega popravljanja nalog. Predlagan sistem je opisan v poglavju **4**. Poglavje **5** vsebuje predstavitev primera ogrodja in testne metode. Poglavje 6 vsebuje zaključek članka.

2 PREDSTAVITEV PODROČJA

V tem poglavju predstavljamo področja, ki so ključna za razumevanje delovanja sistema in so pomembna za njegovo gradnjo oziroma izvajanje. Najprej opredelimo definicijo e-učenja kot ključno kategorijo, v katero se umešča ta sistem. Nato opišemo testno voden razvoj in testiranje enot, ki imata ključno vlogo pri gradnji nalog in njihovem preverjanju. V sistemu platforma Docker poskrbi za izolacijo testov med izvajanjem, sistem Moss pa je uporabljen za preverjanje podobnosti programske kode z namenom preprečevanja plagiatorstva.

2.1 E-učenje

Zgodovinsko gledano ima e-učenje oziroma e-izobraževanje (angl. E-Learning) korenine v učenju na daljavo. Z nastankom interneta je namreč postalo mogoče, da sta pedagog in učenec ločena krajevno in prostorsko, med njima pa poteka komunikacija (E-izobraževanje - Wikipedija, prosta enciklopedija, 2019). Pri tem ne gre samo za deljenje gradiv v elektronski obliki, v smislu izmenjave preprostih elektronskih sporočil, ampak se e-učenje definira kot pedagogika (umetnost in znanost učinkovitega učenja), ki je še dodatno obogatena s sodobno IKT. E-učenje je tako povezava med IKT in učenjem, vodilno mesto v tej povezavi pa pripada pedagogiki. IKT mora biti zanesljiva in dovolj preprosta za uporabo, da ne moti učnega procesa (Nichols, 2007).
Dandanes se v pojem e-učenja vključujejo:

- učenje na daljavo (ni samo v spletni obliki, npr. DVD z naloženo vsebino za učenje),
- mešani načini deljenja virov (npr. spletni viri dopolnjujejo učenje v učilnici),
- spletno učenje (celotna komunikacija poteka izključno preko spleta),
- interaktivnost vsebin (od preprostih povezav na spletne strani do učnih okolij, ki so popolnoma prilagojena uporabniku),
- sistemi za nadzor učenja (z vključenimi možnostmi ocenjevanja) in
- pedagogika.

2.2. Testno voden razvoj

Testno voden razvoj (angl. Test-Driven Development – TDD) je metodologija, ki s pomočjo zelo kratkih razvojnih ciklov omogoča razvoj programskih in informacijskih rešitev, kar jo uvršča med agilno-iterativne procese. Razvoj poteka tako, da se v posameznem ciklu najprej napiše specifična testna metoda (oz. test), šele nato se izvede implementacija funkcionalnosti (oz. dela specifikacij sistema, ki ga razvijamo). Testna metoda tako vodi proces razvoja implementacije funkcionalnosti. Cikel je zaključen, ko se test izvede pravilno, implementacija funkcionalnosti pa je ustrezno vključena v rešitev. Za posamezno funkcionalnost se lahko ponovi več ciklov. Testi morajo biti spisani tako, da jih odlikujejo lastnosti, kot so: neodvisnost, razumljivost in hitrost.

2.3. Testiranje enot

Testiranje enot (angl. Unit testing) (Myers, Sandler, & Badgett, 2011) je metoda testiranja programske kode, pri kateri se testi izvajajo na nivoju posamezne enote (npr. metode, funkcije ali razreda) v programu. Pri pisanju testov je najpogosteje v uporabi t. i. AAA (angl. Arrange-Act-Assert) vzorec, ki predlaga razdelitev testne metode na tri stopnje:

- urejanje (angl. Arrange) – začetna stopnja, kjer se pripravijo vhodni podatki za testirano enoto in določi pričakovan izhod,
- izvajanje (angl. Act) – klic testirane enote s pripravljenimi vhodnimi podatki,

- potrjevanje (angl. Assert) – končna stopnja, kjer se izvede primerjava med dobljenim izhodom iz testirane enote ter predvidenim izhodom iz začetne stopnje.

2.4. Platforma Docker

Platforma Docker (Docker Inc., 2019; Gradišnik & Majer, 2016) je odprtokodna platforma za razvoj aplikacij, njihovo distribucijo in zagon na različnih infrastrukturah z uporabo t. i. zabojnikov. Zabojnik Docker (angl. Docker container) temelji na tehnologiji zabojnikov iz operacijskega sistema Linux in predstavlja nivo abstrakcije na aplikacijskem nivoju. Njegov namen je izolacija programske opreme od izvajalnega sistema. Znotraj zabojnika se zažene slika Docker (angl. Docker image), ki je izvedljiv paket z vsem potrebnim za zagon aplikacije − izvajalno kodo, datotečnim sistemom in vsemi potrebnimi knjižnicami.

2.5. Sistem Moss

Kot pomoč pri odkrivanju plagiatov uporabljamo sistem za določanje podobnosti programske kode, imenovan Moss (Aiken, 2018) (angl. Measure Of Software Similarity). Algoritem, uporabljen za odkrivanje prevar v sistemu Moss, v primerjavi z drugimi podobnimi algoritmi dosega veliko boljše rezultate (Bowyer & Hall, 1999). Sistemu posredujemo seznam programskih kod, ki jih želimo medsebojno primerjati, ta pa nam kot rezultat vrne datoteko HTML s poudarjenimi deli programskih kod, ki nakazujejo podobnost, in odstotke podobnosti.

Dodatna funkcionalnost sistema je možnost odstranjevanja t. i. lažnih ujemanj, saj se iz primerjave odstrani t. i. skupna programska koda (npr. knjižnice, že napisana programska koda itd.). Sistem Moss trenutno podpira več programskih jezikov: 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, a8086 assembly in HCL2.

Ker sistem Moss poda zgolj odstotek ujemanja posameznih programskih kod, ni namenjen popolnoma avtomatiziranemu odkrivanju plagiatorstva, temveč pregledovalcu služi zgolj kot pripomoček.

3 SORODNE IDEJE

Računalniško preverjanje, vrednotenje in ocenjevanje nalog iz programiranja je staro skoraj toliko kot programiranje samo. Sisteme preverjanja nalog iz programiranja delimo glede na (Ihantola, Ahoniemi, Karavirta, & Seppälä, 2010; Romli, Sulaiman, & Zamli, 2010): stopnjo sposobnosti samodejnega preverjanja, število podprtih programskih jezikov, izvajalno okolje preverjanih programov itd. Uporabljeni so na različnih institucijah (Auffarth, López-Sánchez, Campos i Miralles, & Puig, 2008; Edwards, 2003), kamor moramo prišteti tudi aktualne spletne akademije (Massive open online course - Wikipedia, 2019). Primer takšnih so Coursera (Coursera Inc., 2019), edX (edX Inc., 2019), Udemy (Udemy, Inc., 2019), LinkedIn Business (LinkedIn Learning, 2019), Udacity (Udacity, Inc., 2019) itd., ki so namenjene skoraj neomejenim številom udeležencem. Večina spletnih akademij poleg tradicionalnih gradiv (posneta predavanja, branje, množica problemov) ponuja interaktivne tečaje z uporabniškimi forumi, ki podpirajo interakcijo med študenti, učitelji in asistenti, omogočajo pa tudi takojšne povratne informacije za kvize in naloge. Za podrobnejši pregled orodij za samodejno preverjanje nalog vestnemu bralcu svetujemo ogled dodatne literature (Douce, Livingstone, & Orwell, 2005).

V literaturi (Zeller, 2000) je predstavljen sistem Praktomat, ki študentom omogoča medsebojno pregledovanje in ocenjevanje programov z namenom izboljšanja kakovosti ter sloga programske kode. Študent ima po oddani svoji nalogi možnost pridobiti nalogo drugega študenta, ki jo nato pregleda. Po pregledu študent dobi povratno informacijo o oddaji svoje naloge in možnost ponovne, popravljene oddaje. Sistem pregledovanja je neodvisen od ocenjevanja. Možnost plagiatorstva je zmanjšana s poosebljenimi nalogami in samodejnim testiranjem oddanih nalog.

4 PREDSTAVITEV NAŠEGA SISTEMA

Predstavljen sistem nadaljuje delo predstavljeno v (Zemljak, Novak, Čep, & Verber, 2016). Omogoča preverjanje oddaj študentskih nalog z namenom dviga kakovosti študijskega procesa, saj gre za nepristranost (objektivnost preverjanja pravilnosti programske kode), hitrejše preverjanje študentskega razumevanja učne snovi ter poenotene odzive, ki jih študentje dobijo po oddaji naloge. Poenoteni odzivi študentom ponudijo večjo jasnost pri razumevanju lastnih lukenj v znanju, hkrati pa se jim zagotovi hitrejša in bolj strukturirana pomoč pri odpravi morebitnih napak. Prednost sistema je tudi razbremenitev pedagoškega osebja, saj čas preverjanja nalog ne ra-

ste več linearno s količino oddaj. Treba je pa vzeti v zakup, da je začetna priprava naloge za samodejno preverjanje časovno daljša in je potreben razmislek, ali je uporaba takšnega sistema časovno upravičena pri manjšem številu oddanih nalog.

Delovanje sistema za preverjanje študentskih oddaj je razdeljeno v šest faz:

1. začetna priprava naloge,
2. izdelava in objava predloge, vključno z navodili,
3. prenos in razvrščanje oddanih nalog,
4. preverjanje plagiatorstva,
5. obdelava in testiranje nalog,
6. objava rezultatov.

Prva faza je opravljena s strani upravljavcev sistema in vključuje beleženje funkcionalnih zahtev, implementacijo rešitve s pomočjo TDD, ki bo ustrezala zapisanim zahtevam, ter sestavo besedila naloge. V prvi fazi se tekom ciklov TDD-ja (poglavje 4.1): a) ustvari množica testov, ki služijo kot specifikacija naloge, b) tekom preoblikovanj formira končna arhitektura naloge. Pomemben poudarek pri razumevanju pomena prve faze je, da se testi tekom razvoja v tej fazi ne pišejo z namenom validacije oz. preverjanja pravilnosti delovanja sistema, ampak samo kot specifikacija zahtev. Ko je razvoj zaključen in končna arhitektura naloge z vsemi funkcionalnostmi implementirana, se te iste teste (lahko) prekvalificira po uporabi in se jih s pridom uporabi še za validacijo pravilnosti delovanja študentskih oddaj (tako kot je to v našem sistemu − peta faza). Še en pomemben poudarek: testi so namenjeni samo razvijalcem ter jih študentje pri reševanju naloge ne pišejo (razen, če je to njihova želja, oz. opazijo potrebo po testih za namen validacije lastne rešitve).

Rešitev je treba implementirati v dveh programskih jezikih, saj se mora pri opravljanju vaj zaradi dveh različnih študijskih smeri študentom omogočiti možnost izbire programskega jezika (C# ali C++). V praksi to poteka tako, da se najprej s TDD ustvari rešitev v enem programskem jeziku, nato pa sledi ročna preslikava kode še za drugi programski jezik. Pri tem skrbimo za ohranitev strukture razredov, metod, testov in notranjih implementacij.

Druga faza je namenjena pripravi predloge, ki jo dobijo študentje kot osnovo in vsebuje prototipe metod, ki jih morajo implementirati. Predlogi (za vsak programski jezik ena) se vključno z navodili naloge objavita študentom v spletni učilnici. Sledi čas, namenjen reševanju naloge s strani študentov.

Tretja faza je izpeljana po pretečenem roku za oddajo naloge. Sistem iz spletne učilnice samodejno prenese oddane naloge in jih razdeli v ločene mape (vsak študent ima svojo mapo).

Četrta faza je namenjena preverjanju podobnosti oddanih nalog pred testiranjem. V ta namen se uporabi skripta (Algoritem 4.4), ki za preverjanje uporabi sistem Moss. Odziv je datoteka HTML, ki vsebuje seznam parov datotek z odstotki ujemanja podobnosti. Pare datotek s (pre)velikim odstotkom ujemanja pedagoško osebje ročno preveri.

Peta faza je namenjena obdelavi in testiranju nalog. Pred začetkom testiranja se v vsako študentsko mapo dodata dve datoteki: prva vsebuje teste, druga pa je namenjena gradnji sistema (angl. Makefile). Za vsakega študenta se zažene zabojnik Docker (poglavje 4.3), ki ima dostop samo do študentske mape. Znotraj zabojnika se izvede še zadnji korak predpriprave, kjer se oddana naloga prilagodi z datoteko za gradnjo sistema (poglavje 4.4). Sledi začetek testiranja, ki se izvede po principu črne škatle, in sicer po metodi testiranja enot. Enote so metode, ki jih implementirajo študenti. Za posamezno enoto je zapisanih več testov (poglavje 5), katerih povratne informacije se shranijo v t. i. datoteki odzivov. Testiranje se ovrednoti kot uspešno samo v primeru, če je testirana rešitev uspešno prestala vse teste. Sistem ima za izvajanje testov predvideno tudi časovno omejitev, znotraj katere se morajo testi zaključiti. V nasprotnem primeru se testiranje prekine in test se ovrednoti kot neuspešen.

Šesta faza je objava rezultatov. Uporabi se ocenjevalna razpredelnica, katere prenos je mogoč iz spletne učilnice. Razpredelnica vsebuje podatke za vse študente, ki so bili vključeni v nalogo. Za vsakega študenta z oddano nalogo se izpolnita dva podatka: komentar in ocena. Podatka pridobimo iz datoteke odzivov. S komentarjem se študentu podajo povratne informacije testov. Samo ocenjevanje pa poteka binarno – študent lahko pridobi vse točke ali nič točk. Ocena se določi glede na to, ali je študentova rešitev uspešno prestala vse teste ali ne. Dopolnjeno razpredelnico upravljavci sistema na koncu naložijo v spletno učilnico, s čimer se zaključi ocenjevanje naloge.

4.1. Funkcionalne zahteve in priprava testov

Naloge pri posameznih snoveh iz programiranja morajo biti sestavljene z namenom osvajanja točno določenih konceptov. Na primer preverjanje znanja

uporabe podatkovnih struktur (implementiranih v programerskih knjižnicah) zajema tako preverjanje njihovih metod kot tudi širši pomen uporabe le-teh znotraj algoritmov. Ko je znanje, ki ga mora študent osvojiti, dobro opredeljeno, pa se lahko oblikujejo specifikacije naloge ter funkcionalne zahteve, ki jih mora naloga vsebovati. Seznam funkcionalnih zahtev nato uporabimo kot osnovo za pisanje testov s pomočjo TDD.

Celoten postopek (Beck, 2003) razvoja programske kode po TDD zajema naslednje korake:

1. Zapiše se test, namenjen preverjanju majhnega koščka funkcionalnosti naloge, in se ga požene.
2. Test pade.
3. Zapiše se najmanjša možna koda, ki zadošča testu.
4. Izvedejo se vsi do takrat napisani testi (čemur pravimo tudi regresijsko testiranje), ki morajo biti zdaj uspešni.
5. Izvede se preoblikovanje programske kode (angl. Refactoring), ki izboljša njeno notranjo strukturo, vendar ne spremeni obnašanja navzven. Ponovno se izvedejo vsi testi.
6. Če niso izpolnjene vse na začetku zapisane funkcionalne zahteve, ponovimo vse do zdaj zapisane korake s pisanjem novega testa za neimplementirano funkcionalnost. Sicer je razvoj zaključen.

Na tem mestu podajmo praktičen primer testa za implementacijo ene izmed funkcionalnosti naloge:

Študent mora osvojiti znanje uporabe statične podatkovne strukture Tabela. Ena izmed njenih osnovnih metod je metoda pripravi(velikost), ki poskrbi za začetno inicializacijo tabele s podano celoštevilsko vrednostjo velikost, ki jo metoda prejme kot argument. Funkcionalnost na seznamu zahtev za implementacijo je naslednja: »Metoda pripravi(velikost) mora pri prejetju vrednosti velikosti, ki je manjša ali enaka 0, uvodno nastaviti ter vrniti celo-številsko polje ničelne velikosti.«

Test, ki bo izpolnjeval preverjanje te funkcionalnosti, po izvedbi prvih dveh korakov vsebuje naslednjo kodo:

```
// 1. Velikost polja je negativno število
int velikost = -5;
if (Tabela.pripravi(velikost).Length != 0)
    return false;

// 2. Velikost polja je 0
int velikost2 = 0;
if (Tabela.pripravi(velikost2).Length != 0)
    return false;

return true;
```

Algoritem 4.1: **Test za preverjanje pravilnosti implementacije metode pripravi(velikost).**

Implementacija metode *pripravi(velikost)* z izpolnjeno opisano funkcionalno zahtevo po izvedbi tretjega koraka TDD cikla vsebuje:

```
int[] polje = new int[0];
if (velikost <= 0)
       return polje;
```

Algoritem 4.2: **Del kode, s katero bo metoda pripravi(velikost) izpolnjevala funkcionalno zahtevo.**

Ker imamo do zdaj napisan samo en test, lahko četrti korak preskočimo (pri dodanih novih testih pa se bo ta korak vedno izvedel). Sledi peti korak cikla TDD, kjer izvedemo preoblikovanje napisane kode –

odstranitev nepotrebne spremenljivke polje (zmanjša se število vrstic kode, sprostila se je deklaracija spremenljivke pri nespremenjenem delovanju) – ter izvedemo vse teste.

```
if (velikost <= 0)
    return new int[0];
```

Algoritem 4.3: **Del kode, s katerim bo metoda pripravi(velikost) izpolnjevala funkcionalno zahtevo.**

Peti korak cikla (preoblikovanje) je v osnovi izredno pomemben gradnik TDD-razvoja. Na danem primeru je zaradi preprostosti sicer prišlo samo do odstranitev spremenljivke, vendar bi z nadaljevanjem šestega koraka (dodajanje novih zahtev) hitro prišlo do kompleksnejših preoblikovanj, ki bi imele pomemben vpliv na strukturo končne naloge (npr. *pripravi(velikost)* bi postala razredna metoda, spremenila bi se vidnost metode, metoda bi se lahko razdelila na več delov itd.).

4.2. Preverjanje plagiatorstva

Skripta za preverjanje plagiatorstva kot vhod prejme absolutno pot do mape študentskih oddaj, ločenih po programskih jezikih. Za vsak jezik se ustvari seznam nalog, ki se posreduje na strežnik Moss s pomočjo datoteke *moss.pl* (del sistema Moss). Odgovor sistema Moss je datoteka HTML z rezultati.

```
#!/bin/bash
#vhodni parameter ($1) predstavlja pot do oddaj
assignment="$1"
# ustvarimo seznam nalog za C++
parameters=""
for source in $assignment/cpp/*; do
    parameters="$parameters $source"
done
# pošljemo seznam nalog na strežnik Moss
results=$(./moss.pl -l cc $parameters | tail -n 1)
# pridobimo rezultate preverjanja v obliko HTML
wget -O ${assignment}_cpp.html $results 2> /dev/null
# ustvarimo seznam nalog za C#
parameters=""
for source in $assignment/cs/*; do
    parameters="$parameters $source"
done
# pošljemo seznam nalog na strežnik Moss
results=$(./moss.pl -l csharp $parameters | tail -n 1)
# pridobimo rezultate preverjanja v obliko HTML
wget -O ${assignment}_cs.html $results 2> /dev/null
```

Algoritem 4.4: **Skripta za preverjanje plagiatorstva s sistemom Moss.**

4.3. Zabojnik Docker

Zabojnik Docker je v sistemu uporabljen z namenom izolacije testov, zmanjševanja zunanjih vplivov na testiranje ter za varovanje računalnika upravljavca sistema pred morebitnimi zlorabami.

Za zagon zabojnika je bilo treba ustvariti sliko Docker (Algoritem 4.5), kjer je bila kot osnova uporabljena uradna slika okolja Mono (Docker Inc, 2019). Okolje Mono je odprtokodna implementacija .Net ogrodja in omogoča razvoj aplikacij, napisanih v jeziku

C# na operacijskem sistemu Linux. Uradna slika je razširjena z razvojnimi orodji (angl. build-essential). Gradnja slike (dodatne informacije o tem koraku najdete na (Docker Inc., 2019)) se izvede enkrat in objavi na središču Docker (Docker Inc., 2019).

```
#osnovna slika - podpora za C#
FROM mono
#dodatek - podpora za C++ in makefile
RUN apt-get update && apt-get -y install build-essential
```

Algoritem 4.5: **Izdelava slike Docker. Zabojnik se zažene ločeno za vsakega študenta z ukazom:**

Zabojnik se zažene ločeno za vsakega študentra z ukazom:

```
docker run --name Naziv -v \"" + pot + ":/app\" slika bash -c \"cd /app/ && make\""
```

kjer se s parametrom *name* poda ime zabojnika (v našem primeru je to Naziv). Parameter *v* pritrdi mapo s študentsko oddajo (parameter pot) na pot */app/* znotraj zabojnika. S tem se aplikaciji znotraj zabojnika omogoči dostop do vseh datotek v mapi s študentsko oddajo. Parameter *slika* vsebuje povezavo do slike Docker na središču Docker. Na koncu je podan še skriptni ukaz, ki se izvede ob zagonu zabojnika. Ob izvedbi ukaza pride do premika v mapo *app* in zagona skripte za gradnjo sistema.
Zabojnik je po koncu uporabe treba vedno odstraniti, čemur je namenjen ukaz:

```
docker rm -f Naziv
```

4.4. Datoteki za gradnjo sistema

Za zagon testov se uporabljata skripti za gradnjo sistema: C# (Algoritem 4.6) in C++ (Algoritem 4.7). Skripti najprej iz študentskih nalog odstranita vse nepotrebne knjižnice. Sledi zamenjava imena glavnega podprograma iz *main* v *xmain* (podprogram *main* je namreč že vključen v testih). Po zamenjavah se združita datoteka s testi in študentska naloga. Združena programska koda se prevede v izvajalno datoteko, ki se nato zažene. Odzivi testov se shranijo v datoteko *output*.

```
SRCFILE_EXT = cs
SOURCES = $(wildcard *.$(SRCFILE_EXT))
TARGETS = $(SOURCES:.$(SRCFILE_EXT)=.exe)
# Mono.NET prevajalnik
CC = mcs

.PHONY: all
all: $(TARGETS)

%.exe: %.$(SRCFILE_EXT)
    @sed 's/namespace .*$//namespace NalogaTestiranje/g' $< > $*.ccs0
    @sed 's/Main/xmain/g' $*.ccs0 > $*.ccs
    @cat _test/test.cs >> $*.ccs
    @$(CC) _test/TestFramework.cs /out:$*.exe $(*F).ccs /NoWarn:219
    mono ./$*.exe
```

Algoritem 4.6: **Datoteka za gradnjo sistema za C#.**

```
SRCFILE_EXT = cpp
SOURCES = $(wildcard *.$(SRCFILE_EXT))
TARGETS = $(SOURCES:.$(SRCFILE_EXT)=.exe)
#C++ prevajalnik
CC = g++

.PHONY: all
all: $(TARGETS)

%.exe: %.$(SRCFILE_EXT)
    @sed 's/stdafx.h/cstdio/g' $< > $*.cc0
    @sed -i '1i #include <cstdlib>' $*.cc0
    @sed -i -r '/using namespace std/! s/namespace .*([;{])/namespace
NalogaTestiranje\1/g' $*.cc0
    @sed 's/main/xmain/g' $*.cc0 > $*.cc1
    @sed 's/#include "pch.h"//g' $*.cc1 > $*.cc
    @cat  _test/test.cpp >> $(*F).cc
    @$(CC) -o $*.exe $(*F).cc
    ./$*.exe
```

Algoritem 4.7: **Datoteka za gradnjo sistema za C++.**

4.5. Namizna aplikacija

Za lažjo uporabo sistema je bila s programskim ogrodjem Electron (Electron, 2019) razvita prenosljiva namizna aplikacija, ki deluje kot vodič skozi opisan postopek. Aplikacija izvede samodejen prenos nalog, razvrstitev oddaj v ločene mape ter zgoraj opisane skripte.

5 PREDSTAVITEV OGRODJA IN PRIMERA

Preverjanje pravilnosti študentskih oddaj poteka na osnovi vnaprej pripravljenih testov enot. Enote so v našem sistemu metode, ki jih študenti v predlogi implementirajo v skladu s podanimi navodili. Pri uporabi našega testnega ogrodja študentu ni treba dodati niti ene vrstice kode, vendar pa ogrodje študentu vseeno omogoča implementacijo lastnih metod in razredov.

Osnovno ogrodje za testiranje je skupno vsem nalogam in je implementirano v obeh jezikih: C# (Algoritem 5.1) in C++ (Algoritem 5.2). Koda je pri obeh jezikih zelo podobna, s čimer se olajšata sestavljanje nalog in pisanje testov. Glavne komponente ogrodja so: razred *Scenarij* (od sedaj naprej scenarij), metodi za zagon testov in testne metode. V ogrodju je Scenarij razred, ki vsebuje podatek o nazivu testirane enote, kratek opis scenarija ter referenco na testno metodo, ki se bo izvedla. V metodi za zagon testov (metoda *test_vse*) se najprej deklarirajo vsi scenariji, ki se dodajo na seznam. Sledijo zaporedni klici metode *zazeni_test_scenarij*.

```
public class Scenarij{
    public string opis_scenarija;
    public string ime_testirane_enote;
    public System.Func<string, string, bool> referenca;
    public Scenarij(string opis,
        string ime_enote,
        Func<string, string, bool> testna_metoda){
        opis_scenarija = opis;
        ime_testirane_enote = ime_enote;
        referenca = testna_metoda;
    }}
public static bool test_vse (){
    //so vsi testi prestali testiranje?
    bool uspesni = true;
    //seznam testnih scenarijev
    var testniScenariji = new LinkedList<Scenarij>();
    //deklaracija testnega scenarija
    var test_scenarij = new Scenarij(
        "Queue<Karta> zdruzi(Queue<Karta> posl, Queue<Karta> ekon)",
        "Zdruzi prazni vrsti potnikov",
        test_zdruzi_prazni_vrsti_potnikov);
    testniScenariji.AddLast(test_scenarij);
    //dodamo se druge testne scenarije
    //....
    //zagon testov
    foreach (Scenarij trenutni in testniScenariji){
        if (!TestFrameworkClass.zazeni_test_scenarij(
                trenutni.opis_scenarija,
                trenutni.ime_testirane_enote,
                trenutni.referenca))
            uspesni = false;
    }
    return uspesni;}
```

Algoritem 5.1: Razred Scenarij in funkcija za zagon testov v C#.

```
class Scenarij{
    public:
        string opis_scenarija;
        string ime_testirane_enote;
        bool (*referenca)(const string, const string);
        Scenarij(string opis,
            string ime_enote,
            bool (*testna_metoda)(const string, const string)){
            opis_scenarija = opis;
            ime_testirane_enote = ime_enote;
            referenca = testna_metoda;
        }};
bool test_vse() {
    //so vsi testi prestali testiranje?
    bool uspesni = true;
    //seznam testnih scenarijev
    list<Scenarij> testniScenariji;
    //deklaracija testnega scenarija
    Scenarij test_scenarij(
        "Zdruzi prazni vrsti potnikov",
        "queue<Karta> zdruzi(queue<Karta> &posl, queue<Karta> &ekon)",
        test_zdruzi_prazni_vrsti_potnikov);
    testniScenariji.push_back( test_scenarij );
    //dodamo se druge testne scenarije
    //....
    //zagon testov
    for(Scenarij test_scenarij : testniScenariji){
        if (!zazeni_test_scenarij(
            test_scenarij.opis_scenarija,
            test_scenarij.ime_testirane_enote,
            test_scenarij.referenca))
            uspesni = false;
    }
    return uspesni;}
```

Algoritem 5.2: Razred Scenarij in funkcija za zagon testov v C++.

Metoda *zazeni_test_scenarij* (Algoritem 5.3) je del osnovnega ogrodja, njena naloga je izvedba testa s klicem implementirane testne metode. Metoda vrne informacijo o uspešnosti izvedenega testa. Zaradi možnosti napak med izvajanjem (angl. run time errors) je klic metode obdan s prestreznim delom kode (angl. try…catch), ki v primeru napake v datoteko odzivov zapiše poročilo o napaki, ime testirane enote ter opis scenarija, celoten test pa se tudi označi kot neuspešen.

```
public static bool zazeni_test_scenarij(
    string opis_scenarija,
    string ime_metode,
    Func<string, string, bool> testna_metoda){
    bool uspesen = false;
    try { uspesen = testna_metoda(opis_scenarija, ime_metode); }
    catch (Exception e){
        izpisiNapakoVDatotekoOdzivov(opis_scenarija, ime_metode, e);
        uspesen = false;
    }
    return uspesen;
}
```

Algoritem 5.3: **Funkcija za zagon posameznega testa.**

Testna metoda (angl. test case) vsebuje implementacijo testa in se drži vzorca AAA. Najprej se pripravijo vhodni podatki, sledi izvedba testirane enote in na koncu še preverjanje, ali se rezultati enote ujemajo s pričakovanimi rezultati. Vsaka testna metoda sprejme dva parametra: opis scenarija in naziv testirane enote. Ta podatka se v primeru neuspešno izvedenega testa skupaj z razlago zapišeta v datoteko odzivov. Metoda vrne informacijo o uspešnosti testa. Primer testne metode je predstavljen v Algoritem 5.4.

```
private static bool test_zdruzi_prazni_vrsti_potnikov(
        string opis_scenarija,
        string ime_metode){
    //Arrange - urejanje
    bool uspesno = true;
    var poslovni = new Queue<Karta>();
    var ekonomski = new Queue<Karta>();
    var sim = new Simulacija(); //razred v nalogi

    //Act - izvajanje
    var dejanskoZdruzeni = sim.zdruzi(poslovni, ekonomski);

    //Assert - potrjevanje
    if (dejanskoZdruzeni.Count != 0){
        TestFrameworkClass.izpisiNapakoVDatotekoOdzivov(
            opis_scenarija,
            ime_metode,
            "dodala potnika/-e v izhodno vrsto zdruzenih potnikov.");
        uspesno = false; }
    if (poslovni.Count != 0){
        TestFrameworkClass.izpisiNapakoVDatotekoOdzivov(
            opis_scenarija,
            ime_metode,
            "dodala potnika/-e v vhodno vrsto poslovnih potnikov.");
        uspesno = false; }
    if (ekonomski.Count != 0){
        TestFrameworkClass.izpisiNapakoVDatotekoOdzivov(
            opis_scenarija,
            ime_metode,
            "dodala potnika/-e v vhodno vrsto ekonomskih potnikov.");
        uspesno = false; }
    return uspesno;
}
```

Algoritem 5.4: **Primer testne metode.**

6 DISKUSIJA

Predstavljen sistem se od sorodnih razlikuje predvsem v sočasni uporabi dveh programskih jezikov (C# in C++), kar je tudi največja prednost našega sistema. Omogoča objektivno ocenjevanje, kar pa je pri ročnem pregledovanju in testiranju nalog pri več kot 100 študentih in več članih pedagoškega osebja zelo težko doseči. Prav tako se skrajšata čas pregledovanja in testiranja nalog (Tabela 1) ter čas objave rezultatov, ki vsebujejo poenotene odzive o morebitnih napakah. Odzivi študentom povedo, v kateri metodi in pri katerem testu je prišlo do morebitne napake, s čimer lahko pedagoško osebje oziroma študenti hitreje najdejo mesto napake. To jim je v pomoč pri popravljanju naloge pred ponovno oddajo iste naloge.

Slabosti uporabe sistema so: a) podaljšanje časovnega vložka, potrebnega za pripravo naloge, saj mora pedagoško osebje spisati funkcionalne zahteve, implementirati rešitev s pomočjo TDD in sestaviti predlogo, b) testov se ne objavi, tako da morajo študenti testirati lastno kodo, c) binarno ocenjevanje, kjer že en neuspešen test pomeni nič točk pri nalogi in d) sistem je bolj koristen pri predmetih z večjim številom študentov.

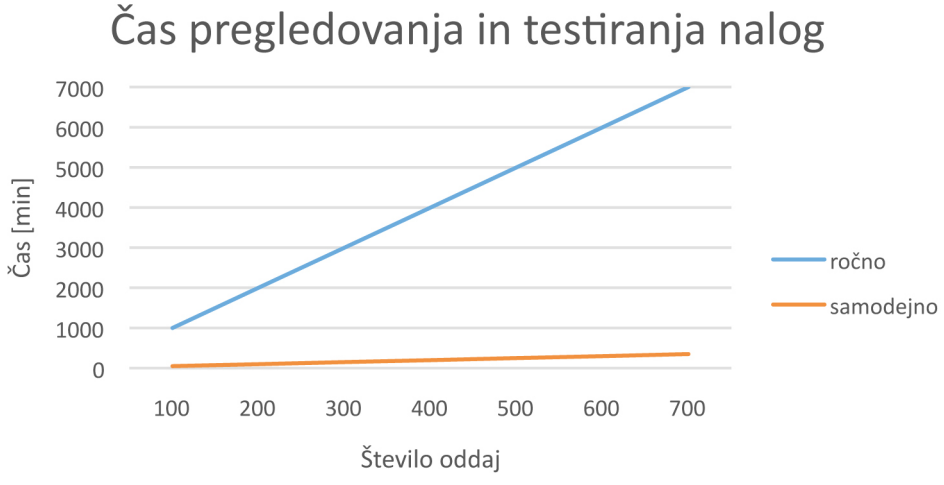
Pri ročnem pregledovanju in testiranju nalog je treba poudariti, da se naloge razlikujejo po težavnosti (Tabela 1), s čimer se čas testiranja poveča. Iz tega razloga smo v Slika 1 za ročni pregled in testiranje nalog uporabili povprečen čas. Za ročni pregled in testiranje ene oddane naloge se glede na naše izkušnje v povprečju porabi približno 10 minut, medtem ko se za samodejni pregled in testiranje ene oddane naloge potrebuje največ 30 sekund.

Tabela 1: **Prikaz težavnosti in časa ročnega pregledovanja in testiranja ene oddaje naloge**

Vrstilec	Stopnja težavnosti	Čas ročnega pregledovanja in testiranja [min]
Naloga 1	4	5
Naloga 2	3	5
Naloga 3	7	10
Naloga 4	8	10
Pregledna naloga	10	20
Povprečje	6,4	10

Slika 1 prikazuje razliko v času med ročnim ter samodejnim pregledovanjem in testiranjem nalog glede na število oddanih nalog, kjer se v povprečju odda 650 nalog na semester. Slika 1 prikazuje potreben čas za pregledovanje in testiranje oddane naloge za enega pedagoškega delavca. Pri samodejnem pregledovanju in testiranju oddane naloge se čas z več pedagoškimi delavci ne bi spremenil, bi se pa porazdelil pri ročnem.

Med študenti smo izvedli tudi krajšo anketo o uporabi našega sistema. Večjih pripomb glede sestave in razumevanj nalog niso imeli. Podali pa so željo, da bi imeli dostop do testov, s čimer bi lahko naloge osebno testirali, preden jih oddajo v ocenjevanje. Kritika sistema je bila izrečena glede poenotenih odzivov o nepravilno delujoči nalogi, saj se študentje niso znali orientirati, kako odpraviti napako (mnenje/utemeljitev avtorjev članka: študentom je v odzivu podana informacija o metodi in krajši opis testa, pri katerem je prišlo do napake, ne pa točna lokacija napake v programski kodi).



Slika 1: **Primerjava časov ročnega in samodejnega pregledovanja in testiranja oddanih nalog**

7 SKLEP

Za razvoj lastnega ogrodja za testiranje enot smo se odločili, ker smo želeli imeti sistem, ki bo notranje podpiral dva programska jezika (C# in C++), navzven pa ponudil identične funkcionalnosti ter uporabniško izkušnjo. S tem se je dosegla lažja integracija implementiranega sistema v študijski proces, pohitrilo pa se je tudi pregledovanje in testiranje nalog. Ena izmed primarnih zahtev, ki smo si jo zadali pri implementaciji sistema, je bila tudi ta, da študentom ne dvignemo kognitivne obremenitve pri osvajanju novega znanja še s prilagajanjem na naše ogrodje. To nam je uspelo, saj študentje ni treba prilagajati svojega stila programiranja predlogi, če tega ne želijo. Jim pa je vseeno prepuščena svoboda v primeru, če želijo dodati novi razredi, strukture, metode itd.

V prihodnosti želimo v sistem dodati tudi funkcionalnosti, ki bi študentom omogočile samostojno rabo celotnega sistema, vendar v okrnjeni obliki (brez vseh testov), kot tudi dodati možnosti za predtestiranja lastnih rešitev za bolj splošne napake (neveljavna sprememba strukture predloge; metoda vsebuje nedovoljene klice iz programskih knjižnic, kot je npr. Array.Sort(); …).

LITERATURA

[1] Aiken, A. (15. 12 2018). *A System for Detecting Software Similarity*. Pridobljeno 9. 9 2019 iz https://theory.stanford.edu/~aiken/moss/

[2] Auffarth, B., López-Sánchez, M., Campos i Miralles, J., & Puig, A. (2008). *System for automated assistance in correction of programming exercises (sac).* Proceedings of CIDUI 2008, (str. 104-113). Lleida.

[3] Beck, K. (2003). *Test-driven development: by example.* Addison-Wesley Professional.

[4] Bowyer, K. W., & Hall, L. O. (1999). *Experience using „MOSS“ to detect cheating on programming assignments.* FIE'99 Frontiers in Education. 29th Annual Frontiers in Education Conference. Designing the Future of Science and Engineering Education. Conference Proceedings (IEEE Cat. No. 99CH37011) (str. 13B3-18). IEEE.

[5] Coursera Inc. (1. 09 2019). *Coursera.* Pridobljeno iz https://www.coursera.org/

[6] Docker Inc. (25. 8 2019). *Mono Docker slika.* Pridobljeno 25. 8 2019 iz Spletna mesto uradne Mono slike: https://hub.docker.com/_/mono/

[7] Docker Inc. (02. 09 2019). *Docker izgradnja slike.* Pridobljeno iz https://docs.docker.com/engine/reference/commandline/image_build/

[8] Docker Inc. (20. 8 2019). *Docker središče.* Pridobljeno 20. 08 2019 iz https://hub.docker.com/

[9] Docker Inc. (10. 9 2019). *Enterprise Container Platform | Docker.* Pridobljeno 25. 8 2019 iz https://www.docker.com/

[10] Douce, C., Livingstone, D., & Orwell, J. (2005). *Automatic test-based assessment of programming: A review*. Journal on Educational Resources in Computing (JERIC), 3.

[11] Edwards, S. H. (2003). *Using test-driven development in the classroom: Providing students with automatic, concrete feedback on performance.* Proceedings of the international conference on education and information systems: technologies and applications EISTA. 3. Citeseer.

[12] edX Inc. (1. 9 2019). *edX | Free Online Courses by Harvard, MIT, & more.* Pridobljeno 1. 9 2019 iz https://www.edx.org/

[13] *E-izobraževanje - Wikipedija, prosta enciklopedija.* (28. 8 2019). Pridobljeno 1. 9 2019 iz https://sl.wikipedia.org/wiki/E-izobra%C5%BEevanje

[14] Electron. (1. 9 2019). *ElectronJS.* Pridobljeno iz https://electronjs.org/

[15] Gradišnik, M., & Majer, Č. (2016). *Mikrostoritve in zabojniki Docker.* V M. Heričko, & K. Kous (Ured.), OTS 2016 Sodobne tehnologije in storitve, (str. 10-20). Maribor.

[16] Ihantola, P., Ahoniemi, T., Karavirta, V., & Seppälä, O. (2010). Review of recent systems for automatic assessment of programming assignments. *Proceedings of the 10th Koli Calling International Conference on Computing Education Research* (str. 86-93). Koli: ACM.

[17] *LinkedIn Learning.* (1. 9 2019). Pridobljeno 1. 09 2019 iz https://www.linkedin.com/learning/topics/business

[18] *Massive open online course - Wikipedia.* (1. 9 2019). Pridobljeno 1. 09 2019 iz https://en.wikipedia.org/wiki/Massive_open_online_course

[19] Myers, G. J., Sandler, C., & Badgett, T. (2011). *The art of software testing.* John Wiley & Sons.

[20] Nichols, M. (2007). No. 1: E-Learning in context. *E-Primer Series, 27.*

[21] Romli, R., Sulaiman, S., & Zamli, K. (2010). Automatic programming assessment and test data generation a review on its approaches. *2010 International Symposium on Information Technology* (str. 1186-1192). IEEE.

[22] Udacity, Inc. (1. 9 2019). *Learn the Latest Tech Skills; Advance Your Career | Udacity.* Pridobljeno 1. 09 2019 iz https://www.udacity.com/

[23] Udemy, Inc. (1. 9 2019). *Online Courses - Learn Anything, On Your Schedule | Udemy.* Pridobljeno 1. 9 2019 iz https://www.udemy.com/

[24] Zeller, A. (2000). Making students read and review code. *ACM SIGCSE Bulletin* (str. 89-92). ACM.

[25] Zemljak, A., Novak, D., Čep, A., & Verber, D. (2016). Preverjanje pravilnosti študentskih nalog programiranja s testiranjem enot. V M. Orel (Ured.), *Sodobni pristopi poučevanja prihajajočih generacij* (str. 556-564). Ljubljana: Polhov Gradec : Eduvision. Pridobljeno iz http://www.eduvision.si/Content/Docs/Zbornik%20prispevkov%20EDUvision_2016_SLO.pdf

▪

Aleš Čep je asistent in doktorski študent na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru. Njegova raziskovalna področja vključujejo umetno inteligenco v igrah ter evolucijsko računanje.

▪

Damijan Novak je leta 2011 diplomiral na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru. Trenutno je asistent in doktorski študent z večjim številom znanstvenih člankov in prispevkov. Njegova raziskovalna področja obsegajo računsko kompleksnost v igralnih svetovih, umetno inteligenco v igrah ter vzpodbujevalno učenje (veja strojnega učenja). Na mednarodni znanstveni konferenci CGAT 2017 mu je bilo za članek "An aspect-based classification of real-time strategy game worlds" podeljeno tudi priznanje za najboljši študentski znanstveni prispevek.

▪

Jani Dugonik je leta 2010 diplomiral in leta 2013 magistriral na Fakulteti za elektrotehniko, računalništvo in informatiko, ki je članica Univerze v Mariboru. Trenutno je doktorski študent in asistent na fakulteti za elektrotehniko, računalništvo in informatiko. Njegova raziskovalna področja vključujejo evolucijsko računanje, optimizacijo, procesiranje naravnega jezika in globoko učenje.

Izpitni centri ECDL

ECDL (European Computer Driving License), ki ga v Sloveniji imenujemo evropsko računalniško spričevalo, je standardni program usposabljanja uporabnikov, ki da zaposlenim potrebno znanje za delo s standardnimi računalniškimi programi na informatiziranem delovnem mestu, delodajalcem pa pomeni dokazilo o usposobljenosti. V Evropi je za uvajanje, usposabljanje in nadzor izvajanja ECDL pooblaščena ustanova ECDL Fundation, v Sloveniji pa je kot član CEPIS (Council of European Professional Informatics) to pravico pridobilo Slovensko društvo INFORMATIKA. V državah Evropske unije so pri uvajanju ECDL močno angažirane srednje in visoke šole, aktivni pa so tudi različni vladni resorji. Posebno pomembno je, da velja spričevalo v 148 državah, ki so vključene v program ECDL. Doslej je bilo v svetu izdanih že več kot 11,6 milijona indeksov, v Sloveniji več kot 17.000, in podeljenih več kot 11.000 spričeval. Za izpitne centre v Sloveniji je usposobljenih osem organizacij, katerih logotipe objavljamo.

AGORA

LJUDSKA UNIVERZA MURSKA SOBOTA

ŠOLSKI CENTER Novo mesto

spin

ISER

LJUDSKA UNIVERZA CELJE
Cankarjeva ulica 1, 3000 Celje, Slovenija
Telefon:(03) 428-67-50 Telefax: (03) 428-67-50

ACADEMIA

Micro Team

Znanstveni prispevki

Kratki znanstveni prispevki

Strokovni prispevki

ISSN 1318-1882
9 771318 188001