Programiranje kao alat za razvoj apstraktnog mišljenja

Nikolina Bubica , prof.
OŠ Mokošica
nikolina.bubica@skole.hr

Monika Mladenović, mad. educ.
PMF Split
monika.mladenovic@pmfst.hr

doc.dr.sc. Ivica Boljat
PMF Split
boljat@pmfst.hr

Sažetak

U ovom radu razmatraju se temeljne postavke problemskog programiranja utemeljene na pregledu relevantne literature te dugogodišnjeg programerskog i nastavnog iskustva autora. U prvom dijelu razmatraju se kriteriji koji se koriste za poučavanje programiranja i računalne znanosti općenito i to kroz povijesnu prizmu pristupa ovoj problematici. U drugom dijelu daje se pregled programskih jezika pogodnih za poučavanje početnika u programiranju. Programiranje se ne promatra kao vještina koju je danas poželjno svladati, već alat pomoću kojeg kod učenika razvijamo apstraktno mišljenje.

Ključne riječi: programiranje, početnici, metakognicija, apstraktno mišljenje, znanje, učenje

Uvod

Programiranje nije samo pisanje računalnih programa. Programiranje je rješavanje problema, otklanjanje grešaka, razvijanje logičkog razmišljanja i računalnog razmišljanja, a to podrazumijeva razvoj strategija za rješavanje problema koji se mogu odnositi i na neprogramerska područja. Zbog toga se može reći da programiranje mijenja način razmišljanja [1]. Kada uzmemo u obzir da živimo u digitalnom dobu i da je djeci tehnologija „prirodno“ okruženje, jasno je da bi takve promjene trebali uzeti u obzir u školovanju [2]. S obzirom na sveprisutnost informatike u svakodnevnom životu, i s pretpostavkom da će ta prisutnost postajati sve izraženija, računalno razmišljanje postaje uz čitanje, pisanje i aritmetiku četvrta „analitička sposobnost“ [3].

Početkom 1970-ih, pojavom osobnih računala, populariziralo se programiranje. Početni programi bili su usko vezani uz računanje i matematiku, a edukacija za programiranje se radila isključivo na sveučilištima. Danas je jedan jednostavan pametan telefon snažniji od svih sveučilišnih računala 1970-ih. Djeca su uključena u računalno okruženje od malih nogu i očekuju interakciju sa računalom pa to područje više nije usko vezano samo uz matematiku i matematičke sposobnosti [3].

Unatoč početnom entuzijazmu za programiranje izraženom 1970-ih i 1980-ih, danas opada zanimanje za učenje programiranja i računalne znanosti općenito, a većina djece u osnovnim školama ima negativno mišljenje o računalnoj znanosti [4]. Neće puno djece postati programeri, a oni koji se odluče za to, svladat će vještine programiranja na fakultetu. Čemu onda djecu učiti programirati?

Programeri znaju da se ne može postati programer samo formalnim obrazovanjem. Svi programeri su morali naučiti učiti, odnosno morali su razviti metakognitivne sposobnosti koje su jako bitne u školovanju [5]. Iz navedenog je jasno kako se u ovom radu programiranje ne promatra izolirano. Programiranje se ne promatra kao vještina koju je danas poželjno svladati, već kao alat pomoću kojeg kod učenika razvijamo metakognitivne sposobnosti. Metakognitivne sposobnosti temelj su uspješnosti u daljem školovanju.

Ostaje pitanje kako danas poučavati ovakve vještine? Gotovo od početka razvoja računalne znanosti nastavnici uočavaju činjenicu da mnogi učenici jednostavno „ne shvaćaju“ neke pojmove, te da postoji velika razlika između najboljih i najlošijih [6]. Zbog toga se razvio velik broj programskih jezika za poučavanje programiranja, od kojih su neki prilagođeni djeci već od predškolske dobi.

U ovom radu razmatraju se temeljne postavke problemskog programiranja utemeljene na pregledu relevantne literature i dugogodišnjeg programerskog i nastavnog iskustva autora. U prvom dijelu se poučavanje programiranja i računalne znanosti razmatra kao alat za razvoj apstraktnog mišljenja općenito i to kroz povijesnu prizmu pristupa ovoj problematici. U drugom dijelu daje se pregled programskih jezika pogodnih za poučavanje početnika u programiranju.

Programiranje i apstraktno mišljenje

Prema Piagetu, postoje četiri faze kognitivnog razvoja čovjeka: senzomotorna (0-2 g.), predoperacijska (2.-7. g), konkretnih operacija (7.-11.g) i formalnih operacija (>12 g) [7]. Odlazak u školu uobičajeno se događa u fazi konkretnih operacija gdje je dijete u stanju logički pristupiti rješavanju problema. Vrijeme ulaska u fazu formalnih operacija varira, dobar dio ljudi nikada ne dođe do ove faze koja je, zapravo, sposobnost apstraktnog mišljenja. Očito je da je ljudima prirodno razmišljati od konkretnog prema apstraktnom.

Papert, kao jedan od tvoraca programskog jezika LOGO, u knjizi *Mindostorms* iz 1980. predstavio je LOGO kao kamen temeljac za pristupe razmišljanja u učenju i

poučavanju. U današnje vrijeme, kada su računala jeftina i svima dostupna, računalo lako može postati alat koji potiče učenje. Kroz svoju knjigu Papert pokazuje kako računalo može doprinijeti mentalnim procesima ne samo instrumentalno već esencijalno, na konceptualnoj razini i to na način na koji ljudi razmišljaju i kada nisu u interakciji s računalima [8].

Papert je bio Piagetov učenik pa je za vrijeme zajedničkog rada u Ženevi imao prilike vidjeti djecu koja su naučila programirati uz pomoć konkretnog modela računala za učenje o učenju (i razmišljanje o razmišljanju) i na taj način ojačala sebe kao psihologe ili epistemiologe. Na primjer, mnoga su djeca nazadovala u učenju jer su smatrala da se može ili „naučiti“ ili „krivo naučiti“. Kod programiranja se, međutim, problem gotovo nikada ne riješi od prve. Učiti za programera znači ovladati višim vještinama izoliranja i ispravljanja pogrešaka („bug-ova“) zbog kojih program ne radi. Pravo pitanje za rad programa nije je li dobar ili nije, nego je li popravljiv. Kada bi se pogled na intelektualne produkte poopćio i primijenio na kulturu razmišljanja o znanju kao akviziciji, smanjili bi strah od neuspjeha, odnosno činjenja pogrešaka. Ovakav mogući utjecaj računala na naše shvaćanje crno-bijelog svijeta primjer je uporabe računala kao „objekta za poticaj razmišljanja“. Naravno da računalo nije neophodno za razvijanje strategija učenja. Uostalom, strategije otklanjanja problema („debagiranja“) su postojale i prije pojave računala, ali analogija između razmišljanja o učenju i razvijanja programa je vrlo moćna i dostupna [8].

Grupa American Association for the Advancement of Science izdala je knjigu *Znanost za sve Amerikance* [9] koja objašnjava razumijevanje znanosti koju bi svi učenici trebali naučiti. Cijelo jedno poglavlje posvećeno je načinu na koji se poučava znanost u pedagoškom smislu. Važna tvrdnja je da učenici uglavnom uče od konkretnog prema apstraktnome. Djeca o svijetu uče uglavnom preko svojih osjetila. Odrastanjem se razvija sposobnost apstraktnog mišljenja, zaključivanje, generaliziranje i sl. Ove sposobnosti se razvijaju sporo i potrebno ih je redovito vježbati. Konkretna iskustva su najučinkovitiji način učenja kada se pojavljuju u kontekstu relevantne konceptualne strukture. Iz ovakvih tvrdnji pojavljuju se tri važna pitanja za računalnu znanost: sveprisutnost računalne znanosti, važnost konteksta te pomak od konkretnog prema apstraktnome [10].

Programiranje je samo po sebi potpuno apstraktno i samim time teško za razumijevanje, ali je u isto vrijeme dobar alat za vježbu i razvoj apstraktnog razmišljanja. Kako djeca na početku osnovne škole, barem većina, nemaju mogućnost apstraktnog mišljenja, učenje programiranja čini im se teškim. Kao odgovor na ovaj prepoznati problem

pojavljuju se vizualni programski jezici. Oni, osim uklanjanja problema sintakse, omogućuju učenje programiranja u konkretnom okruženju, gdje apstraktni pojmovi poput varijabli, petlji i sl. u vizualnom okruženju pružaju konkretno iskustvo. Uz veliki broj programskih jezika koji stoje na raspolaganju, teško je odlučiti se za neki konkretni, pa je potrebno razmotriti različita rješenja.

Učenje programiranja kao kultura učenja

Okruženje poučavanja možemo analizirati i opisati s tri osnovne komponente: učenik, učitelj i sadržaj. Prikažemo li odnose među komponentama Kansanenovim didaktičkim trokutom [11], primijetit ćemo da je u procesu proučavanja početnog programiranja važno uzeti u obzir i kontekst situacije učenja jer se proces učenja ne događa u vakuumu.

Vrlo često učitelji poučavaju programiranje s obzirom na činjenicu kako su oni povezani s računalom, a ne s obzirom na to kako su učenici povezani s računalom [11]. Rasprave o učenju i poučavanju programiranja često se vode bez jasnih poveznica prema učenicima i učiteljima i temelje se na pretpostavci: *što je tehnologija jasnija, lakše se uči*. Kolling, Koch i Rosenberg [12] spominju 10 zahtjeva za prvu godinu učenja nekog programskog jezika, no samo u jednom od tih 10 zahtjeva spominju se učenici. Stalno se vode rasprave o pravom izboru prvog programskog jezika. Mnogi smatraju da učenici imaju problem u učenju jer nisu u stanju vizualizirati rad algoritma. Početkom sedamdesetih godina, rani fenomenografi su prepoznali dva različita pristupa koja koriste učenici pri učenju: dubinski pristup učenju kojim učenici nastoje razviti stvarno razumijevanje onog što uče, te površinski pristup učenju kod kojeg učenici žele jedva izvršiti zadatak koji su dobili od učitelja [11]. Učitelj i sadržaji koje učitelji poučavaju, jedan su od faktora koji bitno utječu na to hoće li učenici prisvojiti površinski ili dubinski pristup učenju [13]. U istraživanjima učeničke motivacije uočena je razlika u intrinzičnoj i ekstrinzičnoj motivaciji, no vrlo često pod intrinzičnom motivacijom promatraju se učenikova intrinzična svojstva neovisno o njegovim reakcijama na učitelja ili sadržaj koji se uči.

Računalni znanstvenici su ili dio ili nositelji određene kulture koja ima svoje vrijednosti i norme koje moraju biti jasno izražene kroz poučavanje. Razlike između kulture učitelja i kulture učenika temeljene su na njihovim iskustvima kućnih računala i igranja. Greške u programima koje učenici rade mogu imati upravo kulturološke razloge jer pojam ispravnosti programa za učenike može predstavljati jedno, a za učitelje potpuno nešto drugo [11].

Programski jezici za početno učenje programiranja

Naglasak u programiranju kod djece mora biti na logici i računalnom razmišljanju. Najveći problem početnih programa za programiranje je složena sintaksa. Ona je često izvor frustracija, neuspjeha i odustajanja unatoč možda točnoj semantičkoj strukturi. Ako govorimo o djeci u prvom razredu osnovne škole, problem je još veći jer djeca u prvom razredu još ne znaju čitati ni pisati. Kako ih onda učiti programirati?

Jedan od prvih programskih jezika koji su imali jednostavnu sintaksu i bili primjereni za djecu, programski je jezik LOGO koji se još uvijek koristi za poučavanje programiranja. Papert, jedan od tvoraca LOGO-a, tvrdio je da programski jezici trebaju imati „nizak pod" (eng. *Low floor* - treba biti lako započeti), „visoki strop" (eng. *High ceiling* - mogućnost da se s vremenom naprave sve kompleksniji projekti), i „široke zidove" (eng. *Wide walls* – mogućnost podržavanja različitih tipova projekata za osobe sa različitim interesima i stilovima učenja). Zadovoljavanje trojke *niski pod/visoki strop/ široki zidovi* nije lako [1]. LOGO je otvorio put novim idejama i po uzoru na LOGO napravljeni su novi alati za vizualno programiranje. Vizualno programiranje se danas dosta koristi za početno učenje koncepata računalne znanosti i programiranja jer podupire učenje putem istraživanja [14]. Prednost takvih alata je u vizualnom okruženju koje dokida potrebu učenja sintakse jezika za razumijevanje i učenje koncepata programiranja [15].

Poučavanje programiranja rješavanjem problema je nažalost donekle zanemareno. Današnji priručnici za programiranje sve češće nalikuju na vodiče kroz neki jezik, s malim naglaskom na rješavanje problema. Kod novih alata za vizualno programiranje, kao što su Scratch, Alice i Greenfoot, vraća se naglasak na rješavanje problema. Učenici mogu programirati u kontekstu. Unatoč činjenici da kontekst igra važnu ulogu u životu učenika, dosta priručnika za programiranje stavlja koncepte programiranja izvan konteksta [16].

Ovakva okruženja, osim pristupa licem u lice (eng. *face-to-face*), pružaju i mogućnost udaljene komunikacije u obliku „galerija", internetskih stranica na kojima učenici mogu postaviti i dijeliti svoje radove, što djeluje poticajno. Takav pristup se naziva i „vrata hladnjaka" (eng. *Fridge door approach* - „na kraju vrtićkog dana djeca vole svoje slike donijeti kući kako bi ih pokazali obitelji i stavili na vrata hladnjaka") [17].

Jedno od okruženja za programiranja koji zadovoljava navedene uvjete je Scratch. Napravljen je 2003. godine na MIT-u od strane grupe „*Life long kindergarden*", koja je

surađivala s tvrtkom Lego na izradi robota Lego Mindstorms [18] koji se, također, koriste za poučavanje programiranja. Uočilo se da djeci u radu sa kockicama odmah počinju navirati ideje, mašta i kreativnost pa su napravili vizualni programski jezik koji podsjeća na slaganje kockica. Naredbe su napravljene u obliku slagalica, tako da je vizualno jasno koje se naredbe mogu složiti. Grupirane su tematski, a razlikuju se po obliku i bojama. Osim toga, pazilo se i na socijalnu komponentu pa Scratch ima veliku Internet zajednicu gdje se mogu dijeliti radovi [1].

Programska okruženja koja se još spominju u kontekstu početnih okruženja za učenje programiranja su Alice i Greenfoot. Navedeni programi imaju dosta zajedničkog sa Scratchom, a razlikuju se s obzirom na dob korisnika za koje su napravljeni. Tako je Scratch prilagođen već za predškolsku dob, Alice za srednju školu i niže godine fakulteta, a Greenfoot za niže godine fakulteta. Oni nisu primarno ni konstruktivistički, ni bihevioristički, ni socijalni. Iako sadrže elemente ovih teorija, ne može se reći da eksplicitno slijede neku od njih. Umjesto toga, može se reći da oni predstavljaju ono što njihovi kreatori vjeruju o učenju programiranja - „spontanu filozofiju“ [19].

Začetnik Alice je Randy Pauch, a napravljena je kao alat za brzo prototipiranje virtualnih svjetova, kao i Scratch, a po uzoru na LOGO. U početku se koristio za više disciplina (računalna znanost, dizajn i umjetnost). Radilo se timski s osobama iz različitih disciplina čime su, osim samog alata, učili i zajednički raditi [10]. Poticanje učenika da uče jedno dok oni misle da rade drugo, Pauch je nazvao „head fake“ (prijevara glave) [20] i upravo u toj činjenici krije se ljepota ovakvog načina poučavanja. Učenici nisu ni svjesni da zapravo sudjeluju u rješavanju problema, otklanjanju pogrešaka, izradi scenarija, u stvari da vježbaju računalno razmišljanje, već misle da rade video igrice, stripove, spotove i sl.

Greenfoot je, poput Scratcha i Alice, programsko razvojno okruženje za učenje i poučavanje početnika. Uzrast za koji je Greenfoot primjeren je od 14 godina pa sve do nižih godina fakulteta. Jedan od osnovnih ciljeva pri izradi Greenfoota je eksplicitna vizualizacija koncepata objektno-orijentiranog programiranja. Iako postoji i uređivač koda, učenici ne započinju sa uređivanjem koda već sa vizualnim naredbama.

Cilj rada sa Greenfootom je upoznati učenike s osnovnim konceptima potrebnim za razumijevanje objektno-orijentiranog programiranja, koji su klasičnim predavanjima teški za razumijevanje jer su vrlo apstraktni. U Greenfootu kod se čini opipljivim s eksplicitnom vizualizacijom i vođenjem korisnika kroz interakciju koja uključuje konkretne događaje [21].

Istraživanja o korištenju programa Scratch

Postoje brojna istraživanja korištenja Scratch, Alice i Grenfoot programa, ali u ovom radu ćemo navesti neke koji se odnose na Scratch zbog nižih uzrasta koji mogu početi programirati.

Provedeno je niz istraživanja o korištenju Scratcha za poučavanje programiranja. U jednom od istraživanja gdje se uspoređivao LOGO i Scratch pokazalo se da je Logo bolji za samopouzdanja učenika, zanimanje za programiranje te razumijevanje petlji. Scratch je pokazao poboljšanje ishoda učenja kod učenja uvjetnog grananja ili ponavljanja. Iako je u Scratchu jednostavnije raditi, začuđujuće je što su učenici, koji su učili Logo i Scratch, bili podjednaki u interpretiranju petlji. Kod Loga se petlja radi tekstualnim naredbama pa se učenici moraju koncentrirati na detalje niske razine. Možda baš to omogućuje učenicima usmjeriti se na važne detalje koje inače zanemaruju i time nadoknađuju nedostatak vizualnog prikaza [22].

Česta su istraživanja u ljetnim školama i klubovima gdje djeca nisu u školskom okruženju i time nisu opterećena ocjenama. U istraživanju tijekom Harvardove ljetne škole računalne znanosti počelo se učiti programirati u Scratchu, a kasnije se prešlo na Javu. Mali broj učenika je imao iskustva u programiranju. Nakon završetka škole učenici su imali izrazito pozitivna iskustva sa Scratchom. Učenici koji su odgovorili da je Scratch negativno utjecao na iskustvo sa Javom, kao razlog su naveli da je Java bitno teža, manje zabavna, a rezultati su siromašniji. Oni koji su neutralno ocijenili iskustvo sa Scratchom, uglavnom su učenici s prethodnim programerskim iskustvom. Takvim učenicima Scratch je bio prejednostavan [23].

Veliko istraživanje provedeno je u siječnju 2005. u tehnološkom centru za izvanškolske aktivnosti u okolici Los Angelesa, trajalo je 18 mjeseci i u njemu je pregledano 536 učeničkih radova. Djeca su uglavnom sama učila jer je cilj ove analize bio proučiti do koje će mjere djeca usvojiti koncepte, kao i izmjeriti je li zajednica kao cjelina povećala znanje o računalnoj znanosti tijekom vremena. Pokazalo se da je ovo jedna od rijetkih programerskih inicijativa u kojima su uspješno sudjelovali dječaci i djevojčice različitih rasa i uzrasta. Ono što je bilo najzanimljivije, je ideja djece o programiranju. Postavljen je niz otvorenih pitanja za bolje razumijevanje kako su djeca doživjela Scratch u odnosu na ostale alate kod kuće, u školi i klubu. Pitalo ih se podsjeća li ih Scratch na barem nešto iz škole, svi su odgovorili da ih podsjeća na barem jedan predmet: umjetnost (n = 20), jezična umjetnost,

posebno čitanje (n =10), matematika (n = 8), znanost (n =5), povijest ili društveni predmeti (n =3), i informatika (n =2). Kada se pitalo dalje o sličnosti s umjetnošću naveli su crtanje ili izradu skulptura (n = 11), dramu (n = 6), glazbu (n=4), i ples (n = 3). Iz većine odgovora uočljivo je da Scratch doživljavaju više kao umjetnost ili osobno izražavanje.

Većina učenika izradu skripta nije doživljavala kao programiranje. Zapravo, kada ih se pitalo što je to programiranje, odgovarali su da ne znaju. Iako ta činjenica u početku zabrinjava, zaključak je da je to dobro jer su upravo zbog toga djeca mogla doživjeti programiranje kao nešto „cool“, a zapravo se dogodila „prijevara glave“ [20]. Na kraju, cilj uključivanja djece u programiranje nije napraviti hakere ili programere, već ih uključiti u veliki raspon tehnologija koje uključuju programiranje, a dio su 21. stoljeća. Ovo je još važnije kada se zna da 90 % djece koja su sudjelovala u istraživanju, nikada nisu imala Informatiku kao predmet tijekom svog školovanja. [24]

Faktori koji najviše utječu na uspjeh u programiranju

Rezultati koje učenici postižu na predmetima programiranja ne koreliraju dobro s njihovim ostalim akademskim rezultatima. Naše razumijevanje tog fenomena još uvijek je nedovoljno. Velik broj istraživanja proučavao je utjecaj faktora kao što su prostorno razmišljanje i matematičke sposobnosti, muzičke sposobnosti, stručnost u govornom jeziku, no to su faktori koji ne izdvajaju programiranje od ostalih disciplina. Opsežnija istraživanja ukazala su da se upravo nivo ugode, matematičko predznanje te atribucija uspjeha mogu smatrati dobrim prediktorima uspjeha u programiranju [25]. Nivo ugode kod učenika ukazuje na to koliko učenici smatraju da je programiranje teško te koliko su pri tome anksiozni. Razina atribucije uspjeha učenika temelji se na njihovim uvjerenjima o razlozima njihovog uspjeha ili neuspjeha kojeg doživljavaju. Rezultati istog istraživanja ukazuju da je ozbiljan pristup učenju u pozitivnom odnosu s uspjehom iz programiranja dok površan odnos prema učenju ima upravo negativan odnos prema istom [26]. Učenicima je važno osjećati da mogu postići uspjeh i napredovati u programiranju. Kako učenici doživljavaju programiranje? Što je po njihovom mišljenju potrebno za uspješno svladavanje zadataka programiranja? Oni su jako svjesni zahtjevnosti ovih zadataka te logičko razmišljanje i sposobnost rješavanja problema smatraju najsnažnijim prediktorima uspjeha u programiranju [26]. Veoma važnim smatraju karakteristike kao što su obraćanje pažnje na detalje, razmatranje alternativa, matematičke sposobnosti i znanje programiranja, a na dnu ljestvice smještaju sposobnost učenja, znanje o računalstvu te modularno inženjerstvo i planiranje. Zanimljivo je da vještinu slušanja i

timskog rada ne smatraju posebno zanimljivim za disciplinu programiranja [26]. Nekoliko različitih istraživanja potvrdilo je povezanost između sposobnosti učenikovog rješavanja problema i izvedbe programiranja [27]. Byrne ukazuje da spol nema utjecaja na izvedbu programiranja [28]. Slično, Allert [29] i Byrne [28] pronalaze da prethodno računalno iskustvo također ne utječe bitno na izvedbu programiranja. Provedeno je više istraživanja koja su proučavala povezanost uspješnosti programiranja i stilova učenja kao što su Kolbovi te Felder-Silvermanovi stilovi učenja. Rezultati u istraživanjima su različiti pa se opći zaključak o povezanosti stila učenja i programiranja ne može izvesti [30].

Teškoća programiranja nije jedini razlog nepopularnosti programiranja kod učenika. Postoji više sociooloških razloga zbog kojih učenici nemaju interes prema ovoj disciplini. Česta je percepcija informatike kao društveno izolirane karijere, informatičke učionice se često doživljavaju kao neosobna muška okruženja kontroliranih ponašanja [31], a sami učenici rijetko vide relevantnost programiranja u stvarnom životu. Potencijal za privlačenje više različitih skupina ljudi na području računalnih znanosti skriva se upravo u stvaranju okruženja koja se bave nekim od tih sociooloških prepreka za programiranje, u podržavanju učenika te u davanju zanimljivih razloga za programiranje [32].

Utjecaj nastavnih metoda i sljedova na percepciju programiranja

Gradivo programiranja učenici ocjenjuju kao veoma teško i često odustaju od njih. Svaki učitelj, pripremajući se za nastavu, pokušava pronaći onaj najbolji put za početnike i često se susreće s dilemom uz primjenu nastavnih metoda. Je li bolje početi sa strateškim ponavljanjem, detaljima sintakse i raditi dalje ili je bolje raditi kroz cijeli koncept programiranja odjednom? Pokazalo se da različitim rasporedom nastavnih sljedova nećemo bitno povećati učinke učenja, no bilo bi pogrešno zaključiti da su nastavni sljedovi potpuno nevažni. Učenici selektivno prihvaćaju informacije koji im se prezentiraju zbog čega je važno birati sljedove upravo sprema spremnosti učenika. Dugoročna memorija učenika može jednostavno ignorirati materijale koji su im prezahtjevni jer oni povećavaju njihovo kognitivno opterećenje. Njihova procjena težine prezentiranog sadržaja i te procjena količine uloženog napora ne mijenja se u ovisnosti o trenutku prezentiranja sadržaja [33]. Učenici će prepoznati težak sadržaj bez obzira na redoslijed njegovog prezentiranja. Ovo ističe važnost razmatranja kognitivnog opterećenja pri određivanju redoslijeda prezentiranja te utjecaju teškog materijala i velikog kognitivnog opterećenja na ukupnu procjenu nastavnog materijala. Pokazalo se da učenje od konkretnog prema apstraktnom proizvodi najmanju ocjenu težine i

najveću učinkovitost. Učenici moraju imati iskustvo s temeljnim elementima programiranja kako bi mogli uspješno shvatiti načine na koje su ti elementi povezani u rješenja. Učitelji bi trebali temeljito raditi na osnovnim vještinama i jednostavnim problemskim rješenjima prije nego krenu sa složenim planiranjem.

Ne možemo reći da postoji dogovor o tome što se pokazalo najefikasnijom metodom u poučavanju programiranja. U većini škola danas je prisutan tradicionalan pristup koji se sastoji od lekcija, domaćih zadataka i možda demonstracija rješenja. Pokazalo se da nije problem naučiti sintaksu ili semantiku pojedinačnih programskih jezika već usvojiti proces kombiniranja tih dijelova u smislene cjeline. Mnogi učenici odustaju jer nisu bili u mogućnosti riješiti zadatke i zbog toga se smatraju nesposobnim za ovo gradivo. Osjećaju se veoma loše i imaju nisku atribuciju uspjeha. Pristupi u učenju koji naglašavaju optimizirano poučavanje te vođenje učenika pozitivno utječu na motivaciju učenika te razinu ugode. Jedan od takvih pristupa je metoda kognitivnog naukovanja koja se temelji na ideji Vygotskog. Vygotski smatra da će učenje biti učinkovitije ako se učeniku daje samo onoliko informacija koliko je potrebno da ojača sposobnost učenika da izvrši zadatak. Ova metoda naglašava učenje radeći i ohrabrivanje učenika na traženje informacija. Vježbanje se smatra obveznim i započinje se veoma rano. Upravo ideja o stalnoj povratnoj informaciji od strane učitelja neiskusnim početnicima pomaže u svladavanju zahtjevnog početka, a učenici to smatraju motivirajućim i nagrađujućim [34].

Učitelju programiranja važno je znati kakve mentalne modele razviju učenici pri usvajanju ključnih programerskih koncepata. Malobrojna istraživanja koja su se bavila upravo mentalnim modelima kod programera početnika ukazala su na njihovu raznolikost, čak i kod veoma jednostavnih koncepata kao što je pridruživanje vrijednosti [35]. Upravljanje ovim različitostima kroz povezivanje i pomaganje učenicima u razvijanju održivih modela iz neodrživih, pravi je izazov za edukatore računalne znanosti. Istraživanja predlažu pristup programiranju s naglašenim konstruktivizmom radije nego objektivizmom. Mnogi učenici razvijaju duboko ukorijenjene, prethodno definirane ideje o konceptima programiranja, kao što je to kod koncepta pridruživanja vrijednosti. Konstruktivistička teorija tvrdi da je tradicionalan pristup poučavanja na temelju nastavnih lekcija i priručnika previše pasivan i da ne radi dovoljno da izazove prethodno definirane ideje te da pomogne učenicima u razvoju održivih mentalnih modela. Umjesto toga, konstruktivizam tvrdi da učenici aktivno stvaraju svoje znanje kombinirajući eksperimentalni rad na postojećim kognitivnim strukturama. Ključna strategija poučavanja temelji se na konstruktivnoj perspektivi kognitivnog konflikta

koja jasno izaziva postojeće ideje kako bi ohrabrila učenike da prepoznaju pogreške u svom razumijevanju te ih natjerala na poboljšanje svojih neodrživih modela. Kognitivni konflikt sam po sebi ne može biti dovoljan za postizanje promjene u neodrživim modelima. Učenik mora biti podržan u procesu kreiranja održivih modela, koncepti moraju biti prezentirani na način i stilom da dozvoljavaju stvarni konstruktivizam ovisnih modela. Programerima početnicima nedostaje neophodna baza znanja za stvaranje održivih modela koncepata programiranja. Vrlo često oni pogrešno koriste svoje prethodno znanje i adoptivne intuitivne modele. Ben-Ari [36] smatra da vizualizacija programa ima potencijal stvoriti prigodno okruženje učenja. Tehnike vizualizacije koriste se preko 20 godina i još uvijek se nisu pokazale uspješnim i učinkovitim koliko se to očekivalo. Zbog toga se predlaže, kao mogući način napredovanja, usvajanje modela temeljenog na kognitivnom konfliktu koji pomaže učenicima shvatiti da postoji problem u trenutnom razumijevanju te korištenje vizualno orijentiranih okruženja učenja za podršku u ispravljanju njihovih neodrživih modela [35].

Zaključak

Tradicionalnim načinom poučavanja programiranja poučavanje se vrši izvan konteksta, izvan realnog svijeta. Sadržaj se može koristiti pri učenju svih koncepata programiranja kroz primjere te ilustracije praktičnih vježbi. Sadržaj mogu učenici poznavati i prije te vidjeti kako će se program u njega uklopiti, što za početnika može biti stimulativno i uzbudljivo. Angažman učenika je ključan kako bi oni nešto naučili dovoljno dobro da bi to mogli negdje ponovo upotrijebiti. Važno je odabrati sadržaj važan za učenike koji može demonstrirati kako se koncepti naučeni u jednom području mogu primijeniti u nekom drugom pa je danas učenje programiranja prisutno kroz mikrosvjetove, robote, programiranje igara, web sadržaj te medijsko računalstvo [27].

Kontekst učenja programiranja olakšava unaprjeđivanje osobnih metakognitivnih sposobnosti te razvoj apstraktnog mišljenja i u područjima izvan programiranja. U tom smjeru namjeravamo provesti istraživanja s učenicima nižih razreda osnovne škole.

Popis literature

1. Resnick, M., i dr. Scratch: Programming for all. *Communications of the ACM.* 2009, Vol. 52, br. 11, str. 60-57.

2. Prensky, M. Digital natives, digital immigrants. *On the Horizon.* 2001, Vol. 9, br. 5, str. 1-6.

3. Wing, J.M. Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences.* 2008, Vol. 366, 1881.

4. Violino, B. Time to Reboot. *Communications of the ACM - A Direct Path to Dependable Software.* 2009, Vol. 52, br. 4, str. 19.

5. Weinberg, G. *The Psychology Of Computer Programming.* New York : Dorset House Pub, 1998.

6. Fincher, S. i Utting, I. Machines for Thinking. *ACM Transactions on Computing Education.* 2010, Vol. 10, br. 4.

7. Piaget, J. *The origins of intelligence in children.* New York : International Universities Press, 1952.

8. Papert, S. *Mindstorms: children, computers, and powerful ideas.* New York : ACM, 1980. 0-465-04627-4.

9. Rutherford, F. J. i Ahlgren, A. *Science for All Americans.* New York : Oxford University Press, 1991.

10. Dann, W. i Cooper, S. Alice 3: Concrete to Abstract. *Commun. ACM.* 2009, Vol. 52, br. 8.

11. Berglund, A. i Lister, R. Introductory programming and the Didactic Triangle. *Twelfth Australasian Computing Education Conference (ACE 2010).* siječanj 2010, Vol. 103.

12. Michael Kolling, Bett Koch, John Rosenberg. *Requirements for a First Year Object-Oriented Teaching Language*. SIGCSE . 3/95, Nashville, 1995.

13. J.B. Biggs. *Teaching for quality learning at university*. Buckingham : Open University Press/Society for Research into Higher Education (second edition), 2003.

14. Meerbaum-Salan, O., Armon, M. i Ben-Ari, M. *Habits of Programming in Scratch.* Darmstadt, Germany : ACM, 2011. ITiCSE '11 Proceedings of the 16th annual joint conference on Innovation and technology in computer science education. str. 168-172. 978-1-4503-0697-3.

15. Ward, B., i dr. *Teaching computer science concepts in Scratch and Alice*. Journal of Computing Sciences in Colleges. 2010, Vol. 26, br. 2, str. 173-180.

16. Cooper, S. The Design of Alice. *ACM Transactions on Computing Education.* 2010, Vol. 10, br. 4, str. 1-16.

17. Fincher, S., i dr. *Comparing Alice, Greenfoot & Scratch.* New York : ACM, 2010. Proceedings of the 41st ACM technical symposium on Computer science education. str. 192-193. 978-1-4503-0006-3.

18. Resnick, M. Behavior construction kits. *Communications of the ACM - Special issue on computer augmented environments: back to the real world.* 1993, Vol. 36, br. 7, str. 64-71.

19. Fincher i Utting. Machines for Thinking. *ACM Transactions on Computing Education* . 2010, Vol. 10, br. 4.

20. Pausch, R. i Zaslow, J. *The Last Lecture.* New York : Hyperion, 2008.

21. Kölling, M. The Greenfoot Programming Environment. *ACM Transactions on Computing Education.* 2010, Vol. 10, br. 4.

22. Lewis, C.M. *How programming environment shapes perception, learning and goals: logo vs. scratch.* New York : ACM, 2010. Proceedings of the 41st ACM technical symposium on Computer science education. 978-1-4503-0006-3.

23. Malan, D.J. i Leitner, H. H. Scratch for budding computer scientists. *ACM SIGCSE Bulletin.* 2007, Vol. 39, br. 1, str. 223-227.

24. Maloney, J.H., i dr. Programming by choice: urban youth learning programming with scratch. *ACM SIGCSE Bulletin - SIGCSE 08.* 2008, Vol. 40, br. 1, str. 367-371.

25. B.C. Wilson, S. Shrock. Contributing to success in an introductory computer science course: a study of twelve factors. *ACM SIGCSE Bulletin*. 33, 2001, Vol. 1, str: 184-188.

26. Simon, i dr. Predictors of Success in a First Programming Course. *Australian Computing Education Conference (ACE).* 2006, Vol. 52, str. 189-196 .

27. de Raadt, M. Introductory Programming in a Web Context. *12th Australian Computing Educational Conference (ACE).* 2010, Vol. 103, str. 79-86 .

28. G. Byrne, M. Martin, L. Staehr. Computer Attitudes and Computer Career Perceptions of First Year Computing Students. Krakow, Poland : *Proceedings of Informing Science 2001 - Bridging Diverse Disciplines, e-Proceedings, 2001*. ISSN 1535-0703.

29. J. Allert. Learning style and factors Contributing to Success in an Introductory Computer Science Course. Washington, USA : *ICALT '04 Proceedings of the IEEE*, stranice 385-389, 2004. ISBN: 0-7695-2181-9.

30. Pillay, N. i Jugoo, V.R. An investigation into student characteristics affecting novice programming performance, Vol. 37 br. 4. *ACM SIGCSE Bulletin.* 2005, Vol. 37, br. 4, str. 107-110.

31. K. Garvin-Doxas, L.J. Barker. Communication in computer science classrooms: understanding defensive climates as a means of behaviors. New York : *ACM Journal on educational Resources in Computing (JERIC)*, Vol. 4, br. 1, 2004.

32. Kelleher, C. i Pausch, R. Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers. *ACM Computing Surveys.* 2005, Vol. 37, br. 2, str. 83.-137.

33. Kranch, D. A. Teaching the novice programmer: A study of instructional sequences and percetion. *Education and Information Technologies.* 2012, Vol. 17, br. 3, str. 291-313.

34. Vihavainen, A., Paksula, M. i Luukkainen, M. Extreme Apprenticeship Method in Teaching Programming for Beginners. *SIGCSE '11.* 2011, str. 93-98.

35. Ma, L., i dr. *Investigating the viability of mental models held by novice programmers.* 2007. SIGCSE '07 Proceedings of the 38th SIGCSE technical symposium on Computer science education. str. 499-503 .

36. M. Ben-Ari. Program Visualization in theory and practice. *Informatique*. Vol. 2, str 8-11.