Strategije poučavanja i faktori koji utječu na unapređenje znanja programera početnika

Nikolina Bubica
nikolina.bubica@du.t-com.hr

***Sažetak*: Istraživanja o izazovima poučavanja programera početnika dugo su već predmet interesa svih faktora uključenih u izučavanje uvodnog predmeta programiranja. Programeri početnici često se trude prebrzo shvatiti što je to programiranje. Takav pristup obično je površan, a može dovesti do frustracija i odustajanja. Problem se očituje kroz niske prolazne ocjene na uvodnom predmetu programiranja te visok postotak odustajanja od tog predmeta na raznim fakultetima. Cilj ovog rada je dati pregled relevantne literature i istraživanja o strategijama poučavanja programera početnika koje su se pokazale uspješnima. Navode se faktori koji mogu značajno utjecati na uspjeh u učenju programiranja te iznose iskustva o utjecaju izbora prvog programskog jezika na ukupan uspjeh učenika u uvodnom predmetu programiranja.**

***Ključne riječi:* prediktori, početnici, programiranje, programski jezici, strategije poučavanja**

I. UVOD

Tijekom posljednja dva desetljeća došlo je do duboke promjene u načinu na koji se percipira i provodi učenje unutar sustava obrazovanja. Poučavanje sve više postavlja učenika u središte procesa učenja umjesto učitelja koji dominira u tradicionalnom pristupu poučavanju. Kod tradicionalnog poučavanja računarske znanosti učenici imaju velike poteškoće pri usvajanju novih koncepata [1]. Učitelji moraju prilagoditi svoj pristup poučavanju kako bi olakšali početnicima usvajanje zahtjevnih koncepata programiranja te ih potaknuli na veći angažman u uvodnom predmetu programiranja.

U ovom radu rasprava o poželjnim strategijama poučavanja te prikladnim okruženjima i programskim jezicima odnosi se isključivo na programere početnike. U radu je korištena relevantna literatura iz područja istraživanja računarske znanosti, ali i mnoštvo istraživanja koja se bave navedenim temama. Pri odabiru istraživanja namjera je bila uključiti i ona istraživanja koja ukazuju na neke nove trendove i ideje koje je potrebno detaljnije istražiti [2].

Prema definiciji koju su postavili Bonar i Soloway programera početnika možemo opisati kao nekog tko je u prvoj fazi procesa nastajanja programera, a prema Smith-u kao krajnjeg korisnika koji želi programirati računalo. Početnici se pojavljuju u rasponu od onih koji nikada prije nisu programirali pa sve do onih za koje se može reći da imaju neka osnovna znanja o programiranju, prikupljena putem neformalnog, informalnog ili formalnog poučavanja. Iako takve učenike smatramo početnicima u programiranju, većina njih ima neko iskustvo i znanje o računalima koje je uglavnom neujednačeno.

Računarska znanost ima važan zadatak prepoznati ono što je učenicima zaista zanimljivo dok programiraju, otkriti kontekst u kojem to učenici rade te kako to rade. Najranija razumijevanja programiranja smještaju programiranje dijelova koda u žarište učenja. Takvo programiranje započinjalo je uvijek s nekoliko linija koda, a završavalo gotovom aplikacijom čija svrha nije bila neka primjena nego usvojenost programerskih koncepata. Učenje programiranja predstavljalo je praksu koja izuzetno cijeni preciznost i učinkovitost. Danas, učenici umjesto da programiraju samo zato da bi programirali, radije kreiraju stvarne aplikacije kao dio veće zajednice učenja npr. igre i priče [3]. Igre, sofisticiranih animacija te posebno nijansirane digitalne priče nude jednostavan ulazak u zajednice programiranja i njeguju „mrežno prijateljstvo“ [4]. Najnovija promjena u shvaćanju programiranja obilježena je korištenjem i miješanjem gotovih dijelova programa tzv. remixa što predstavlja zaštitni znak novih mrežnih zajednica programiranja. Od nekadašnjeg dominantnog individualističkog pristupa svjedoci smo doživljavanja računalnog razmišljanja kao **računalnog sudjelovanja** [3] (**engl. computational participation)**.

Novi pristup poučavanju u posljednjem desetljeću ogleda se kroz nastavne metode koje proizlaze iz teorije konstruktivizma. Konstruktivizam je teorija učenja koja tvrdi da učenici konstruiraju znanje, a ne samo primaju i pohranjuju znanje preneseno od strane nastavnika. Ova teorija tvrdi da je znanje aktivno izgrađeno od strane učenika, a ne pasivno apsorbirano iz udžbenika i predavanja. Izgradnja se događa rekurzivno nad postojećim znanjem koje učenik već posjeduje, a to su činjenice, ideje i uvjerenja. Svaki učenik konstruira svoju verziju znanja. Takve nastavne tehnike trebale bi biti uspješnije od tradicionalnih tehnika, jer neminovno vode prema izgradnji znanja [5]. Ben-Ari [5] ističe da učenik početnik nema učinkovit model računala. Učinkovitim modelom računala smatra se svaka kognitivna struktura koju

Rad je u sklopu Istraživačkog seminara I predan na ocjenu 19. travnja 2014. povjerenstvu: prof. dr. sc. Marko Rosić, predsjednik, prof. dr. sc. Andrina Granić, doc. dr. sc. Ivica Boljat, mentor

Nikolina Bubica student je poslijediplomskog doktorskog studija Istraživanje u edukaciji u području prirodnih i tehničkih znanosti, usmjerenja informatika na Prirodoslovno-matematičkom fakultetu

učenik može koristiti kako bi stvorio održive konstrukcije znanja. Takve konstrukcije nastaju na temelju osjetilnih iskustava kao što su čitanje, slušanje, predavanja te rad na računalu. Zadatak učenika početnika je izgraditi od nule ideju o tome kako radi računalo. U programiranju, konkretno, ta izgradnja povezana je sa sposobnošću predviđanja i razumijevanja onog što se događa pri izvođenju određenog programa. Standardne strategije poučavanja nisu uspjele riješiti niz poteškoća s kojima se susreću programeri početnici pri izgradnji učinkovitog modela računalnog programa. Istraživanja uspjeha učenika na uvodnom predmetu programiranja pokazuju njihove loše rezultate u odnosu na rezultate u ostalim predmetima. Upoznat ćemo neke od strategija koje su pokazale potencijal u radu s početnicima. Uz pregled programskih jezika i okruženja za učenje navedeni su kriteriji koji mogu pomoći učitelju u izboru odgovarajućeg programskog jezika za početnike. Pri izboru odgovarajućeg načina rada s početnicima učitelj mora voditi računa i o faktorima koji mogu utjecati na unapređenje njihovog znanja. U posljednjem dijelu rada dan je pregled dobrih prediktora uspjeha u programiranju. Navode se i faktori koji do sada nisu pokazali snagu predviđanja ili njihov utjecaj na izvedbu programiranja nije dovoljno istražen.

II. STRATEGIJE POUČAVANJA PROGRAMERA POČETNIKA

Konstruktivistička epistemologija slijedi kognitivnu pretpostavku da znanje promatramo kao strukturu povezanih informacija umreženu s iskustvom i stalnom reorganizacijom. Kako pojedinci postaju više iskusni u određenim znanjima oni proizvode sofisticiranije i složenije sustave. Stručnjaci su skloniji stvaranju generalizacija, dok početnici mogu samo napraviti površno promatranje [6]. Koncepti programiranja koje učenici usvajaju razlikuju se svojom težinom od jednostavnijih kao što su varijable, strukture grananja i strukture petlji, zatim rada s ulaznim i izlaznim podacima, manipulacijom pogreškama pa sve do zahtjevnijih koncepata polja, strukturiranih tipova podataka, rekurzija, pokazivača i referenci [7]. U cilju usvajanja novih znanja, učenici moraju ugraditi nove informacije u postojeće sheme te ih prilagoditi istima [8, 9]. Učitelji bi trebali uskladiti svoju nastavnu praksu s načinom na koji učenici percipiraju zahtjevne pojmove programiranja, a ne s načinom na koji su učitelji sami povezani s računalom. To nije u skladu s konstruktivističkim pristupom koji smješta učenika u središte procesa odnosno zagovara poučavanje s obzirom na način na koji su učenici povezani s računalom. Svako se učenje može shvatiti kao ulazak u neku kulturu. Računalni znanstvenici nositelji su kulture ili barem jedan dio određene kulture koja ima svoje vrijednosti i norme jasno izražene kroz nastavu. S druge strane, učenik uvodnog predmeta programiranja nosi sa sobom određena iskustva kućnih računala i igranja na računalu pa upravo tu možemo tražiti razlike među kulturama učitelja i učenika [10, 11]. Višenacionalno istraživanje programerskih sposobnosti početnika provedeno 2001. godine ukazalo je da učitelji imaju siromašno znanje o onom što učenici zaista nauče na uvodnom predmetu programiranja te da slabo poznaju probleme s kojima se učenici susreću pri učenju programiranja [12]. Ohrabrujuće djeluju rezultati istovrsnog novijeg istraživanja provedenog 2013. godine koje je ukazalo da učitelji danas ipak bolje poznaju svoje učenike, ali i da je došlo do promjena u načinu poučavanja, okruženjima učenja te u samim učenicima [13].

Na koji način programeri početnici usvajaju teške koncepte uvodnog predmeta programiranja? Koje strategije primjenjuju za rješavanje problema koji se temelje na teškim konceptima? Još sedamdesetih godina prošlog stoljeća rani fenomenografi uočili su dva različita pristupa učenju koja koriste učenici. Dubinski pristup učenju u kojem učenici teže razvijanju pravog razumijevanje onoga što uče te površinski pristup učenju u kojem učenici jedva žele obaviti zadatak kojeg su dobili od učitelja. Može li učitelj pozitivno djelovati na učenika kako bi razvio kvalitetan pristup učenju? Učitelji i sadržaj učenja snažno utječu na izbor dubinskog ili površinskog pristupa učenju kod učenika [14, 15]. Iako su se mnoge studije bavile temom poučavanja programera početnika ne postoji dogovor o tome koja se metoda pokazala najuspješnijom. Fincher [6] analizira interaktivne pristupe kao što su: pristup **bez-sintakse (engl. syntax-free**), pristup **računalne pismenosti** (engl. **computational literacy**), pristup **rješavanja problema** (engl. **problem-solving**) i pristup **računarstvo kao interakcija** (engl. **computing as interaction**). Zajednički izazov ovih pristupa je u napuštanju stajališta da bi učenici trebali učiti kod te iz tog iskustva naučiti složene, prenosive vještine kao što su analiza, dizajn i rješavanje problema. Navedeni pristupi zagovaraju učenje koda odvojeno od učenja programiranja. Razlike među pristupima očituju se kroz njihovu primjenjivost u učionicama. Pristupi računalne pismenosti i rješavanja problema gotovo da ne zahtijevaju izmjene postojećeg kurikuluma te više ovise o stavovima, vještinama i ciljevima učenja učitelja. Pristupi bez-sintakse te računarstvo kao interakcija zahtijevaju znatno veće prilagodbe postojećih nastavnih materijala. Ovi pristupi prepoznaju stjecanje nekih drugih vještina kao krajnji cilj učenja te ističu važnost podrške učenicima u postizanju tog cilja.

Većina sveučilišta i dalje rade na tradicionalan način u uvodnom predmetu programiranja. Tradicionalni format poučavanja sastoji se od predavanja, zadataka i možda demonstracija pojedinih zadataka. Lekcije se uglavnom bave dijelovima programskog jezika umjesto nešto općenitijim strategijama same primjene programiranja. Nije toliki problem naučiti sintaksu ili pojedinačne semantike programskih jezika već usvajanje i kombiniranje tih komponenti u smislenu cjelinu. Dijelovi programskih jezika o kojima se govori u lekcijama, obično se koriste i u procesu vježbanja. Uz vrlo malo podrške u procesu programiranja, rješavanje problema može biti vrlo teško za dio učeničke populacije, osobito one koju u literaturi nazivamo neučinkovitim početnicima [16]. Mnogi od njih odustali su od svojih predmeta, jer nisu bili u stanju riješiti zadatke, osjećali su se neadekvatnim i nesposobnim za taj zadatak. Pojavljuje se i problem s pisanjem domaćih zadaća. Samostalno rješavanje domaće zadaće smatra se okruženjem s minimalnom mogućom pomoći što je u psihologiji poznato kao pristup koji nije optimalan za početnike pri rješavanju kognitivno zahtjevnih zadataka. Zadaci programiranja definitivno spadaju u kategoriju kognitivno zahtjevnih zadataka pa se postavlja pitanje u kojoj mjeri je dobro učenicima zadavati takve zadatke. Može se dogoditi da učenici stvore loše navike

pri samostalnom rješavanju zadatka te kao posljedicu toga razviju loše kognitivne strukture znanja. I na sveučilišnoj razini obrazovanja, koja od studenta očekuje dosta samostalnog rada, predviđa se mogućnost konzultacija kao pomoći u radu.

U cilju razbijanja prakse „biti poučavan“, koja se često primjećuje i prije osnovnog obrazovanja, uvodni predmeti programiranja na nekim sveučilištima organizirani su na način da naglašavaju angažman učenika umjesto pasivnog učenja. Jedna od strategija koja njeguje takav pristup je **kognitivno naukovanje** (engl. **Cognitive Apprenticeship**) koje stavlja naglasak na usvajanju kognitivnih vještina. Kognitivno naukovanje dijeli poučavanje u tri faze:

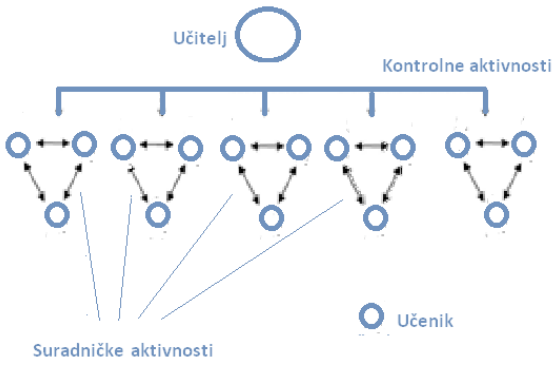
- **modeliranje** (engl. **modeling**),
- **skaliranje** (engl. **scaffolding)** i
- **nestajanje** (engl.**fading)**.

Učitelj u fazi modeliranja pruža studentima konceptualni model procesa kojim stručnjak obavlja promatrani zadatak. Učinkovit način modeliranja može percipirati poučavanje kroz radne primjere. Radni primjer prikazuje izradu zadatka programiranja od početka do kraja. Učitelj cijelo vrijeme glasno razmišlja, objašnjavajući odluke donesene u postupku rješavanja. Nakon faze modeliranja prelazi se u fazu skaliranja u kojoj su studenti rješavaju zadatak pod nadzorom učitelja stručnjaka. Izuzetno je važan način pomaganja studentima. Osnovna ideja je ne pružati im jasne odgovore već samo naznake i kratke informacije uz pomoć kojih mogu jednostavno doći do odgovora na pitanje. Kako studenti stječu vještine, skaliranje se napušta čime počinje faza nestajanja. Ranija istraživanja pokazala su da kognitivno naukovanje, gdje početnicima informacije daju stručnjaci, donosi višu stopu zadržavanja učenika u početnim predmetima računarskne znanosti [17]. Strategija **ekstremnog naukovanja** (engl. **Extreme Apprenticeship**) [18, 19] naglašava komunikaciju između nastavnika i učenika za vrijeme učenikovog procesa rješavanja problema. Takav pristup čini angažman učenika korisnim i smislenim za sve sudionike obrazovnog procesa [18, 20]. Naglašava se učenje radeći, te važnost stalne povratne informacije od strane učitelja. Učenike se rano uvodi u rješavanje zadataka pri čemu se zadaci uvijek dijele na manje dijelove. Takvi razlomljeni zadaci učenicima djeluju rješivi i razumljivi. Metoda ekstremnog naukovanja u nastavi programiranja može se integrirati i kao dio **programiranja u paru** (engl.**pair programming)**. Učenici programiraju u paru sa stalnim revidiranjem pisanog koda. Programiranje u paru, softverska razvojna tehnika u kojoj dva programera rade zajedno na jednoj tipkovnici, pokazala je značajne pozitivne rezultate osobito na polju učenikovog osjećaja zadovoljstva i samopouzdanja [21, 22]. Postoji duga tradicija korištenja učenika i studenata preddiplomskog studija, kao asistenata u nastavi općenito. Uključivanje studenata u nastavnu zajednicu što je ranije moguće, korisno je i plodonosno za sve strane u obrazovnom procesu. Studenti početnici imaju koristi od dobro organizirane strukture nastavnog procesa, dok asistenti usavršavaju svoje vještine u nastavi te ostale vještine vezane uz programiranje. S obzirom na očekivanja od studenata, jasno je da su studenti učili i sudjelovali u nastavi [20] odnosno postignuta je visoka razina angažmana studenata.

Uspješnost u poboljšanju stope prolaznosti predmeta uvodnog programiranja te zadržavanje na tom istom predmetu pokazala je metoda **vršnjačkog poučavanja** (engl. **Peer Instruction).** Vršnjačko poučavanje je pedagoška praksa dizajnirana za podršku učeničkog angažmana u nastavi i poboljšavanje ishoda učenja. Ova nastavna praksa započinje određenim brojem pitanja na koja učenici odgovaraju pojedinačno. Nakon pojedinačnih odgovora slijedi manja grupna rasprava te na kraju rasprava koju vodi učitelj s cijelim razredom. Istraživanja primjene ove strategije u nastavi računarske znanosti dala su pozitivne rezultate. Ukazala su na visoko zadovoljstvo učenika, poboljšane rezultate ispita, te više stope prolaznosti ispita uvodnog programiranja [23]. Studentska razmišljanja o strategiji vršnjačkog poučavanja izuzetno su pozitivna, gotovo 91% studenata preporuča svojim profesorima veću uporabu ove strategije u nastavi [23, 22].

Mnoga sveučilišta i škole nemaju dovoljno nastavnog kadra kako bi pomogli svim onim učenicima kojima je pomoć potrebna. Rješenje takvog problema neka sveučilišta vide u primjeni **suradničkog učenja** (engl.**collaborative learning**) među učenicima. Učenici se dijele u manje skupine u kojima zajedno programiraju. U svakom trenutku učenici mogu zatražiti pomoć ili savjet od neke druge skupine učenika. Glasno raspravljaju o rješenju i svako rješenje koje učenici dožive ispravnim i uvjerljivim, učeniku donosi nagradne bodove (slika 1).

Suradničko učenje kombinira učenike u grupe prema raznim kriterijima i karakteristikama učenika. Model koji zagovara suradničko učenje povezujući učenike prema njihovom znanju programiranja i angažmanu u skupini, pokazao se uspješnim u unapređenju znanja učenika bez obzira na neke njihove karakteristike i navike učenja [21].



Slika 1. Model suradničkog učenja.

Za odabir strategija poučavanja učitelju je iznimno važno poznavati način na koji programeri početnici usvajaju teške koncepte. Iskusni programer stvara kod kako bi riješio problem primjenjujući strategije koje su proizašle iz rješavanja nekih prošlih problema, što nije slučaj kod programera početnika [9, 24]. Oni mogu veoma dobro razumjeti sintaktička pravila programskog jezika te napisati jednostavne programe, ali vrlo često teško pronalaze rješenja problema ako nemaju iskustvo jednog eksperta. Poznavanje razlika na koji način početnici i

eksperti analiziraju i rješavaju probleme može pomoći učiteljima. Te razlike istaknut će koncepte na koje se početnici fokusiraju. Pomoći će učiteljima da ne očekuju od učenika ono što u tom trenutku nisu u stanju dati. Učitelji su često u dilemi kako organizirati sam sadržaj učenja programiranja. Je li bolje početi sa strateškim ponavljanjem, detaljima sintakse i tek nakon toga raditi dalje? Ili će bolje rezultate postići učenje svih koncepata programiranja odjednom? Istraživanja koja su se bavila upravo utjecajem rasporeda sadržaja učenja, odnosno rasporeda nastavnih sljedova ukazala su da promjena rasporeda nastavnih sljedova neće bitno utjecati na ono što su početnici naučili. Međutim, ne možemo reći da mijenjanje rasporeda nije bitno. Izmjena rasporeda sadržaja učenja utječe na kognitivno opterećenje i osjećaj težine koji su učenici prepoznali učeći te sljedove [9]. Razlikujemo tri osnovna pristupa u organizaciji nastavnih sljedova. Gagne (1988) predlaže nastavni slijed **elementi u planove (**engl**.elements-to-plans**) koji se temelji na hijerarhijskim vještinama u kojem su sve preduvjetne vještine prikazane i prethodno svladane prije vještina koje o njima ovise. Reigheluth i Stein's (1983) predlažu **elaboracijski slijed,** elaboracijsku teoriju koja se bazira na sadržaju i cjelovitom pristupu koji zagovara učenje od početka svih znanja i vještina potrebnih za izvršavanje zadataka, umjesto individualnog uvođenja koncepata. Soloway (1982) predlaže vođenje početnika prirodnim logičkim jezikom pakiranjem znanja eksperata kao standardnih rješenja ili planova. Problemi se dijele na male dijelove koji odgovaraju planovima. Na kraju je potrebno odrediti koji planovi odgovaraju kojem dijelu te kombinirati planove u jedinstvena razumljiva rješenja. Takav nastavni slijed naziva se **planovi u elemente** (engl.plans-to-elements). Uspoređivanjem ovih nastavnih sljedova pokazalo se da postižu jednake ishode učenja, no redoslijed nastavnih sljedova nije nevažan [9]. Učeničko samovrednovanje težine pojedinih nastavnih sljedova te kognitivnog opterećenja pri usvajanju tih nastavnih sljedova ne mijenja se njihovim razmještanjem [9, 24]. Učenici će prepoznati teške koncepte bez obzira kojim ih redoslijedom prezentiramo. Učenje od osnovnog prema apstraktnom proizvelo je najnižu ocjenu težine te najveću učinkovitost. Učitelji bi trebali temeljito raditi na osnovnim vještinama i jednostavnim problemskim rješenjima prije nego krenu sa složenim planiranjem. Na taj način početnici imaju mogućnost steći potrebno predznanje za razumijevanje i primjenu traženih planova.

Usvajanje vještine programiranja zahtijeva od učenika dugotrajan rad. Većina predmeta uvodnog programiranja dijeli zajednički pristup poučavanju rješavanja problema. Od učenika se očekuje da uče iz svog uspjeha i neuspjeha kako bi razvili visoko strukturirane strategije rješavanja problema i tako bili u mogućnosti rješavati neke buduće probleme. No, takav pristup ima očigledan nedostatak. Početnici ulažu ogroman broj sati u vježbanje, a može se dogoditi da neki učenici razviju neodgovarajuće vještine. Također, pristup često rezultira odustajanjem od predmeta jer uspjeh značajno ovisi o motivaciji učenika. U novije vrijeme sve se više razmišlja o strategijama koje će omogućiti učenicima izgradnju planova bez velikog broja sati vježbanja. Okvir koji se temelji na Soloway-ovom **Cilj/Plan** pristupu u vizualnom zapisu donosi detaljan postupak za korištenje postojećih planova i izgradnju novih planova u vizualnom okruženju programiranja. Iako je ovo još eksperimentalni pristup, rezultati su pokazali značajan potencijal u poboljšanju nastave za početnike [25]. Neka novija istraživanja predlažu uključivanje eksplicitnog poučavanja rješavanja problema temeljeno na **Cilj/Plan** [24] okruženju nudeći to napisano na papiru ili predlažu uporabu uzoraka programa u vođenju početnika kroz probleme raslojavanja [26]. Upravo nedostatak sposobnosti rješavanja problema kod učenika promatra se kao vodeći uzrok neuspjeha u učenju programiranja [27].

Još jedan od potencijalnih uzroka neuspjeha vidi se u stvaranju neodrživih mentalnih modela koncepata programiranja koji mogu dovesti do zabluda i teškoća u rješavanju problema [5]. Što su mentalni modeli? Craik (1943.) prvi predlaže da ljudi prezentiraju svijet koji ih okružuje mentalnim modelima koji se mogu stvarati percepcijom, imaginacijom te konstruktivnim raspravama. Prema Craik-u mozak stvara „modele stvarnosti malog opsega“ koji se mogu koristiti za predviđanje i razumijevanje događaja te podržavanje objašnjenja tih događaja. Prema Normanu (1983) mentalni modeli zadovoljavaju tri osnovna uvjeta:

- Mentalni modeli odražavaju uvjerenja osobe koja ih posjeduje o promatranom sustavu
- Sustav se može promatrati – postoji određena razmjena komunikacija između parametara i stanja mentalnog modela te zadanog sustava koju osoba može promatrati
- Sustav ima snagu predviđanja – svrha mentalnog modela je podržavanje ljudi u razumijevanju i predviđanju ponašanja promatranog sustava.

Daljnja istraživanja mentalnih modela prepoznala su neke njihove zajedničke karakteristike neovisno o konceptima na koje se odnose:

- Mentalni modeli su nedovršeni i pojednostavljeni – učenik ograničen svojim prethodnim znanjem ne može stvoriti cjelovit mentalni model koji pokriva sve detalje promatranog sustava.
- Mentalni modeli su promjenjivi – mijenjaju se i razvijaju kako njihovi vlasnici ostvaruju interakcije sa sustavom.
- Postoji određeno vrijeme kašnjenja u mijenjanju mentalnih modela – pri izmjeni modela stara informacija zamjenjuje se novom, ali to se ne događa trenutno.
- Stara informacija zadržava se još neko vrijeme u memoriji zajedno s novom informacijom.
- Mentalni modeli imaju nejasne granice – slični modeli mogu se miješati i ispreplitati.
- Mentalni modeli su neznanstveni i često sadrže predrasude – ljudi često zadržavaju svoja ponašanja čak i kad znaju da su pogrešna, a uglavnom to rade jer se zbog toga osjećaju ugodno i sigurno.
- Mentalni modeli su oskudni – ljudi teže izbjeći složenost u mentalnim modelima.

Istraživanje mentalnih modela učenika veoma je važno, osobito za pripremanje učitelja te dizajniranje nastavnih materijala. Iako postoji mnogo istraživanja koja se bave ljudskim mentalnim modelima prirodnih fenomena koji ih okružuju,

postoji jako malo istraživanja koja su proučavala upravo mentalne modele programera početnika. U nastavi programiranja učitelju je veoma važno prepoznati mentalne modele temeljnih programerskih koncepata koje posjeduju njegovi učenici. Učiti programiranje uključuje stvaranje održivih mentalnih modela temeljnih koncepata programiranja. Učenici s održivim mentalnim modelima znatno bolje rješavaju zadatke programiranja od onih s neodrživim mentalnim modelima [11] [28]. Za poboljšanje učeničkih mentalnih modela predlaže se pristup poučavanju programiranja koji naglašava konstruktivizam, a ne objektivizam. Vrijedi pretpostavka da mnogi učenici, prije uključivanja u predmet uvodnog programiranja imaju već duboko ukorijenjene ideje o nekim računalnim konceptima kao što je npr. naredba pridruživanja. Teorija konstruktivizma tvrdi da je tradicionalan pristup nastavi, temeljen na predavanjima i udžbenicima, previše pasivan i ne čini dovoljno da izazove postojeće ideje i pomogne učenicima u stvaranju održivih mentalnih modela [5] [11]. Konstruktivizam zagovara aktivno konstruiranje znanja učenika kombiniranjem iskustvenog svijeta s postojećim kognitivnim strukturama.

Strategija poučavanja koja izričito izaziva postojeće ideje upravo je Festinger-ova strategija **kognitivnog konflikta.** Primjena ove strategije izaziva učenika da prepozna pogreške u svom razumijevanju. Time upravo pokušava promovirati poboljšanje neodrživih modela [11]. Strategija kognitivnog konflikta jedna je od ključnih strategija poučavanja koja se temelji na konstruktivizmu, a razvijala se na pretpostavci da učenikovo predznanje i postojeća uvjerenja utječu na način na koji učenici usvajaju nova znanja i grade nove kognitivne strukture. Kognitivni konflikt je stanje u kojem učenik opaža raskorak između njegove kognitivne strukture i vanjskih uvjeta ili između dijelova njegove kognitivne strukture. Model učenja koji integrira strategiju kognitivnog konflikta s vizualizacijom algoritama pokazao se učinkovit za poboljšanje interesa i angažmana učenika u radu s nastavnim materijalima te kao pomoć u izgradnji održivih mentalnih modela [11] [28].

Nešto drugačiji početak programiranja od svih do sada navedenih strategija nudi strategija **izravnog uključenja** (engl. **kick-start activation**) [29]. U ovom pristupu ulazi se u duboku strukturu programiranja prije uvođenja same strukture programskog jezika. Namijenjena je upravo učenicima koji nemaju nikakvih prethodnih iskustava s programiranjem i za njenu primjenu zahtijeva ispunjenost triju kriterija. Osnovni kriterij ove strategije je da se temelji na stvarnom računalnom programu. Koncept algoritma uvodi se kroz pseudo kod i dijagrame čime istovremeno učenici ulaze u rješavanje problema, u faze programiranja te razumijevanje razlike u načinu razmišljanja čovjeka i računala. Drugi kriterij ove strategije zahtijeva jednostavnost. Koriste se primjeri iz svakodnevnog života, bez uporabe stvarnog programiranja u nekom programskom jeziku. Treći kriterij predstavlja zahtjev prema učenicima da sudjeluju u rješavanju primjera što se može izvesti stvaranjem pogrešne verzije algoritma koju učenici moraju popraviti. Uporaba ove strategije zajedno s nekim vizualnim alatom za testiranje i dizajn npr. JHAVE ponudila je pozitivnu povratnu informaciju od učitelja i učenika [29].

U raspravi o strategijama poučavanja važno je spomenuti uporabu vizualizacije za koju mnogi učitelji računarske znanosti vjeruju da ima snažan edukacijski utjecaj. Upotreba alata vizualizacije u nastavi uvodnog predmeta programiranja bila je predmetom mnogih istraživanja. Do sada se vizualizacija nije pokazala učinkovitom u skladu s očekivanjima [30], ali u kombinaciji s nekim zanimljivim načinom poučavanja može dati pozitivne rezultate. Model poučavanja koji integrira strategiju kognitivnog konflikta te vizualizaciju programa kroz alat Jeliot pokazala je dobre rezultate [11, 31]. Sustavna meta-studija 24 eksperimentalna istraživanja vizualizacije algoritama [32] pokazala je da najveći utjecaj na učinkovitost učenja ima upravo način na koji učenici koriste tehnologiju vizualizacije algoritma, a ne ono što su učenici vidjeli u tom procesu. Istraživanje snažno naglašava da je najuspješniji obrazovni put korištenja audio vizualnih tehnologija upravo onaj u kojem se tehnologija koristi kao sredstvo za aktivno sudjelovanje učenika u procesu učenja.

III. PROGRAMSKI JEZICI I OKRUŽENJA ZA POUČAVANJE PROGRAMERA POČETNIKA

A. Pregled programskih jezika i okruženja za poučavanje programera početnika

Valjana, pouzdana analiza problema programera početnika može imati brojne primjene. Na temelju takve analize može se podesiti količina i vrsta pažnje koju učitelji posvećuje različitim temama na predavanjima, laboratorijskim vježbama i u nastavnim materijalima. Razumijevanjem procesa učenja prvog programskog jezika možemo stvoriti učinkovitija okruženja učenja. Većina sustava koji se koriste za poučavanje programiranja bave se samim procesom programiranja. Naglasak je obično na riječima koje naređuju računalu (naredbe) ili na razumijevanju radnji koje se izvode na računalu. Neki drugi sustavi smještaju programiranje u pristupačan i zabavan kontekst poput projekata pripovijedanja, igara i robota [33]. U svjetlu tog razmišljanja pojavili su se sustavi koji pojednostavljuju programske jezike (QBasic, SP/J, Turing, Blue, JJ). Pojednostavljivanje programskog jezika provelo se zadržavanjem samo nekoliko jednostavnih naredbi, smanjivanjem elemenata sintakse ili održavanjem najveće moguće sličnosti s jezicima opće namjene bez mijenjanja strukture naredbi.

Još jednu vrstu sustava predstavljaju okruženja za početnike pristupačna za uređivanje (MacGnome GNOME,..). Ti sustavi nisu bili pravi programski jezici, već okruženja koja su imala osnovnu namjenu spriječiti sintaktičke pogreške početnika. Unatoč pokušajima da se programski jezici naprave lakšim i razumljivijim, mnogi početnici i dalje imaju problema sa sintaksom, pa se razvoj sustava, koji se temelje na pretpostavci da je potrebno u potpunosti zaobići sintaktičke pogreške, nastavlja u tri smjera:

- sustave za kreiranje objekata koji predstavljaju kod, a mogu se pomicati i kombinirati na razne načine
- sustave koji koriste razne akcije u okruženju za definiranje problema
- sustave koji pružaju više načina za kreiranje programa.

Sustavi Play, Show and Tell, Make Believe Castle, Logo Blocks, Alice2, Kodu primjeri su sustava koji razvijaju program korištenjem objekata te grafičkim prikazivanjem svih akcija i događaja. LegoSheet and Tortis primjeri su sustava koji koriste fizičke objekte ili simulacije fizičkih objekata. Akcije korisničkog sučelja mapirane su na naredbe programskog jezika te omogućuju pogrešku ukoliko redoslijed radnji ne odgovara određenoj naredbi. Nešto složeniji način pisanja programa nudi sustav kao što je Leogo koji omogućuje studentima učenje jednostavnih metoda u naprednom načinu rada. Stvaraju se crteži poput logotipa kornjače te kombiniraju tri načina kretanja u programu.

Iskušavanje nekih novih paradigmi programiranja ili mijenjanje postojećih poslužilo je kao temelj za razvoj sustava za poučavanje programiranja putem nekih novih programskih modela ili modela koji postojeće paradigme čine dostupnima. Programski modeli koji uključuju jezike kao što su Pascal, SmallTalk, Playground while LiveWorld, BlueEnvironment, Karel J Robot predstavljaju sustave koji imaju za cilj pojačati dostupnost programiranja. Među svim ovim okruženjima i modelima potrebno je istaknuti one koji nastoje programiranje napraviti što preciznijim i manje apstraktnim. Takvi sustavi su Karel J Robot i Turingal. U skladu s novijim istraživanjima, posebna pažnja posvećuje se angažmanu učenika pa imamo sustave koji promoviraju kolektivno učenje, kao što su AlgoBlocks, te sustave koji pružaju razloge za postupno uvođenje programa za početnike kroz zanimljive teme kao što to nudi RockyBoots.

Tradicionalnim pristupom poučavanju programiranje se uči izvan konteksta, bez dodira sa stvarnim svijetom. Mnoga istraživanja upućuju na prednosti korištenja neke teme kroz predmete uvodnog programiranja. Najbolji primjeri upravo su programi koji uključuje **mikro-svjetove**, **robote**, **igre** te **medijsko računarstvo**. Kontekst se može koristiti tijekom uvodnog predmeta programiranja kroz primjere, ilustracije i praktične vježbe [34]. Studenti mogu biti upoznati s njim i prije, te vidjeti kako se program uklapa u ono što može biti poticajno i uzbudljivo za početnike. Angažman učenika neophodan je kako bi naučio nešto dovoljno dobro i primijenio to negdje drugo. U isto vrijeme, učenje programiranja uz neki kontekst može biti štetan. Ako se nešto naučilo u jednom području, onda se može dogoditi da se taj sadržaj veže isključivo za to područje i ne vidi mogućnost primjene tog znanja negdje drugo. Upravo iz tog razloga pri odabiru konteksta treba birati onaj koji može pokazati kako se koncepti naučeni u jednom području mogu primijeniti u nekom drugom području [34] [35]. Najraniji primjeri uporabe mikro-svjetova u programiranju dizajnirani su u Logu koji se temelji na "fizičkim primjerima" geometrijskih principa, u programiranom grafičkom okruženju, sintakse slične Lispu. Mikro-svjetovi su korisni za nastavu, ali postoji opasnost da učenici neće biti u mogućnosti prenijeti važnost ovog sadržaja u realni svijet. Korištenje robota kao sadržaja u početnom programiranju pokazalo se uspješnim i motivirajućim u mnogim institucijama [36]. Prvo veliko istraživanje [37] pokazalo je negativne rezultate, prvenstveno zbog nedovoljnog broja računala koje su učenici imali na raspolaganju. Kasnije, dodavanjem kamere i pojedinačnih robota svakom učeniku, proizašli su izuzetno pozitivni i ohrabrujući rezultati [38]**.**

Medijsko računarstvo (engl. **media computing**) aktivnost je koja uključuje manipuliranje medijima kao što su slike i zvučne datoteke. Takve aktivnosti potiču kreativno izražavanje, istovremeno baveći se programskim značajkama kao što su petlje i upravljanje podacima. Pokazalo se da se upravo takvim pristupom može postići povećanje pamćenja i entuzijazma među učenicima te potaknuti žene na sudjelovanje. Uspjeh ovog sadržaja utjecao je na njegovu široku uporabu [39].

Zbog raznih poteškoća u učenju programiranja nastavnici razvijaju okruženja koja učenicima olakšavaju taj proces što je više moguće. Mnoga okruženja nisu dovoljno istražena, a samo neka od njih napravila su značajan iskorak u radu s početnicima. Među njima posebno se ističu Alice, Greenfoot i Scratch okruženja. Iako su osmišljena u različitim vremenima intervalima i u različitim kontekstima, ta okruženja možemo zajedno promatrati jer dijele zajednička svojstva. Sva tri okruženja su vizualna, imaju za cilj potaknuti izravno sudjelovanje u nekoj atraktivnoj aktivnosti te sva tri imaju zadatak upoznati učenike s programiranjem. Također, korištenje ovih okruženja, bilo pojedinačno ili u **face-to-face** grupama, podržavaju udaljene zajednice u obliku **galerija**. Galerije su web stranica na kojima učenici objavljuju i dijele svoje materijale. Ovaj pristup zove se pristup **vrata hladnjaka** (engl. **fridge door**). Okruženje Alice je trodimenzionalna animacija interaktivnog svijeta. Programer početnik gradi 3D animirane filmove i vlastite igre dok istovremeno uči osnovne koncepte objektno-orijentiranog programiranja. Uporaba ovog okruženja zaista povećava samopouzdanje studenata. Nepostojanje sintaktičkih pogrešaka povećava sigurnost početnika, no može otežati prijelaz na tekstualne programske jezike [40] kao što su JAVA ili C++. Problem prijelaza na programski jezik JAVA uspješno je riješen u grafičkom okruženju Alice3 [41]. Greenfoot je visoko specijalizirano obrazovno okruženje za razvoj interaktivnih grafičkih aplikacija koje se temelji na programskom jeziku Java. Scratch je medijski vrlo bogato mrežno okruženje koje se temelji na medijima proizašlim iz nekih stvarnih situacija. Ono ističe upravljanje medijima i podržava programske aktivnosti koje su zanimljive mladima, kao što su stvaranje animiranih priča, igara i interaktivnih prezentacija. Uz Scratch važno je spomenuti i Snap, nekadašnji BYOB, vizualni **povuci-pusti (engl. drag and drop)** programski jezik, koji predstavlja proširenu implementaciju Scratcha i omogućuje izgradnju vlastitih blokova. To ga čini pogodnim za ozbiljan uvod u računarsku znanost za srednjoškolce i studente.
U svim ovim okruženjima programiranje se temelji na stvaranju blokova i veoma je zanimljivo za učenike. Lako je povjerovati da takva okruženja mogu angažirati učenike na neko vrijeme, ali važno je znati koliko koncepata programiranja učenici zaista uče, dok stvaraju filmove, igre i priče. Kvantitativna analiza 300 samostalnih učeničkih projekata koji su kreirani u Alice i Scratch okruženjima pokazala su da ta okruženja zaista motiviraju učenike na korištenje raznih stvari. Učenici koji su stvarali igre uglavnom su koristili varijable, petlje i uvjetne izjave, dok je kod projekata s video uradcima uporaba uvjetnih izjava daleko niža [42]. Ovi žanrovi jasno ukazuju što motivira

učenike na njihovo korištenje. Veliku sličnost s navedenim grafičkim okruženjima ima programski jezik Kodu jer je vizualan i dozvoljava stvaranje vlastitih animacija, igara i simulacija. Kodu je programski jezik koji se temelji na programiranju objekata koje je potpuno određeno događajima (engl.event-driven) dok okruženja Alice, Scratch i Greenfoot potpuno slijede sintaksu Java programskog jezika. Korisnici ovog jezika mogu likove pojedinačno programirati, pri čemu se programiranjem likova određuje način komunikacije lika sa svijetom, baš kao inteligentnog agenta. Uporaba Kodu jezika za usvajanje programerskih koncepata u radu s početnicima rezultirala je povećanim angažmanom učenika te učeničkim projektima koji su sadržavali i zahtjevne programerske koncepte [43]. To ukazuje na mogućnost uporabe ovog programskog jezika ne samo kao početnog programskog jezika već i kao prijelaz na nešto složenije programske jezike.

Još jedan zanimljiv pristup početnom programiranju je putem web-sadržaja. Korištenje web sadržaja ima mnoge prednosti u odnosu na tradicionalne metode tijekom uvodnog predmeta programiranja. Web-sadržaj se uklapa veoma dobro u okruženje za učenje, nudi mnogo bolju interakciju od ostalih objekata, a može se koristiti i na različitim platformama koje uključuju web-preglednik i uređivač teksta. Web-sadržaj podržava različite paradigme poučavanja na različitim razinama. Učenici su dobro upoznati s web-sadržajem i cijene priliku za eksperimentiranje s kodom uz posebno pripremljene primjere. Uporaba web-sadržaja sigurno neće značajno smanjiti napor učenika, ali može ih potaknuti na uključenje te angažman na dulji vremenski period [35].

U raspravi o programskim jezicima i okruženjima za učenje početnika važno je razmotriti pitanje odgovarajućeg programskog jezika kojim se uvode učenici u svijet programiranja.

B. Izbor prvog jezika za programiranje

Izbor jezika koji se koristi u uvodnom predmetu programiranja uzrok je dugotrajnih rasprava, često na temelju emotivnih razmišljanja. Trenutno postoji približno 8000 [44] poznatih programskih jezika koji su dokumentirani na World Wide Webu. Koji bi od tih jezika trebao upoznati učenike s odgovarajućim konceptima programiranja na najbolji način te zadržati interes učenika prema programiranju kao bitnom dijelu računarske znanosti?

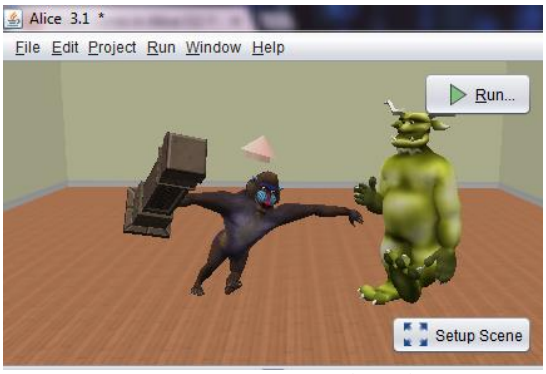
Provedeno je nekoliko istraživanja o prednostima pojedinih jezika te usporedbe pojedinih programskih jezika, ali još uvijek postoji nedostatak objektivnih podataka kojima bi objasnili ove usporedbe. Postoji toliko knjiga, mrežnih sadržaja, primjera kodova koji čak nude učenje programskog jezika „za jedan dan“. Upravo mnoštvo literature koja se bavi istraživanjem ovog pitanja ukazuje da prvi programski jezik ima značajan utjecaj na programera. On utječe na stil programiranja, tehniku kodiranja, kvalitetu napisanog koda na različite načine [45]. U razvoj ranijih programskih jezika, kao što su LOGO, GRAIL i BASIC, bili su uključeni stručnjaci u području programiranja i kao takvi možda nisu uspješno udovoljili pedagoškim zahtjevima programera početnika s obzirom na poteškoće s kojima se programeri susreću. Osnovni cilj svakog uvodnog predmeta programiranja, a time i učenja prvog programskog jezika mora biti usvajanje osnovnih koncepata programiranja koji se mogu primijeniti jednako dobro u bilo kojem programskom jeziku. Također, programski jezik mora biti dovoljno intuitivan za početnika kako on ne bi odustao na samom početku [46]. Gupta [46] predlaže mnoštvo faktora o kojima bi trebalo razmisliti pri izboru programskog jezika kao što su: zahtjev da jezik treba udovoljiti svojstvu *jednostavnosti*, zahtjev za udovoljavanjem svojstva *ortogonalnosti* koje očekuje da ne postoji previše konstrukata s istim funkcijama, zahtjev za *usklađenošću sa ciljanom publikom, pokrivenost svih semantičkih i sintaktičkih konstrukata*, zahtjev za udovoljavanjem svojstva *regularnosti* koje očekuje od programskog jezika semantičku i sintaktičku konzistentnost, zatim zahtjevi koji se odnose na *složenost* i *tipove podataka* koje nudi programski jezik. Osamdesetih godinama prošlog stoljeća usporedbe programskih jezika COBOLa, FORTRANa, Pascala, PL-a, te Snobola, prvenstveno su bile usmjerene na učinkovitost izrade koda te brzinu provedbe koda, a sve kako bi osigurali nastavnicima informacije potrebne za odabir odgovarajućeg programskog jezika.

Danas se razmatranja programskih jezika više fokusiraju na pedagoški aspekt učenja programiranja pa je raspon promatranih jezika i širi. Predloženo je nekoliko različitih kriterija za odabir prvog jezika programiranja, no najsnažniji utjecaj na rad u ovom području došao je od kreatora četiri programska jezika: Seymoura Paperta (LOGO), Niklausa Wirtha (Pascal), Guida van Rossuma (Python) i Bertranda Meyera (Eiffel) [47]. Njihovi kriteriji odnose se na jezike u cjelini bez spominjanja paradigme. Unutar kriterija provjeravaju

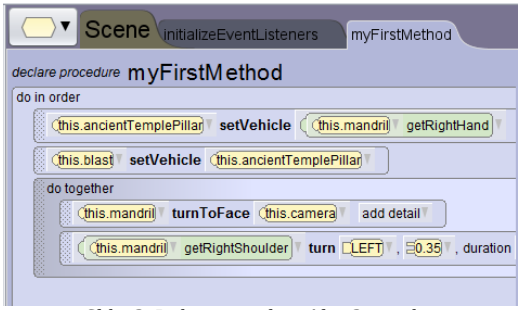
- je li jezik *pogodan za nastavu*,
- nudi li *opći okvir*,
- promovira li *novi pristup* poučavanju programiranja,
- promovira li *pisanje ispravnih programa*,
- dopušta li *probleme koji se rješavaju u "malim dijelovima"*,
- nudi li *fleksibilna razvojna okruženja*,
- nudi li *podršku u obliku zajednice korištenja*,
- je li *otvorenog koda* tako da svatko može pridonijeti njegovom razvoju,
- je li *dosljedno podržan kroz razvojno okruženje*,
- prate li ga *dobri nastavni materijali*
- *koristi li se samo u obrazovanju*.

Iako je bogatstvo i širina kriterija uvelike olakšala snalaženje u odabiru programskih jezika osnovni nedostatak ove analize je u tome što samo provjerava je li određeni kriterij ispunjen ili ne. Analiza ne bilježi u kojoj je mjeri kriterij ispunjen. Između 11 programskih jezika kao što su C, C++, Python, Java, Eiffel, Haskell, JavaScript, Logo, Pascal, VB rezultati istraživanja sugeriraju da su najprikladniji jezici za nastavu upravo Python i Eiffel. Takav zaključak obrazložen je stavom da su ti jezici dizajnirani kao da su namijenjeni poučavanju. Ipak, ne smije se zanemariti važnost i snaga Java programskog jezika, koji je prvenstveno namijenjen za komercijalne aplikacije, a prema ispunjenosti kriterija istraživanja dolazi odmah nakon Pythona i Eiffela. Velik nedostatak Java programskog jezika ipak leži u činjenici da nije namijenjen poučavanju.

Bez obzira na prednosti i prikladnost pojedinih jezika, sintaksa programskog jezika ostaje značajna prepreka programerima početnicima. U analizi početničkih pogrešaka sintaktičke pogreške dolaze odmah nakon nerazumijevanja apstrakcije, polimorfizma i koncepata objektno orijentiranog programiranja [48]. Učiteljima uvodnih predmeta programiranja važno je upoznati razlike u sintaksama programskih jezika za početnike te koliko one utječu na učinkovitost učenja [49].



Slika 2. Projekt u Alice3 okruženju



Slika 3. Prikaz procedura Alice3 projekta

```
private void initializeEventListeners()
    this.addSceneActivationListener( (
        this.myFirstMethod();
    } );
    this.addSceneActivationListener( (
    } );
}
```

Slika 4. Java kod projekta nastao transferom iz Alice3 okruženja

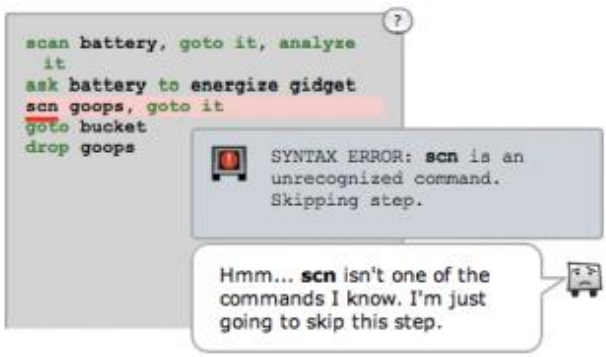
Programiranje u grafičkim okruženjima u kojima ne postoje sintaktičke pogreške može ublažiti stresan početak učenja programiranja. No, sintaktičke pogreške pojavljivati će se i dalje pri prijelazu na neki tekstualni programski jezik. Okruženja kao što su Alice3 nude metode koje izravno pretvaraju Alice3 objekte u Java kod [41] što je od velike pomoći i veoma motivirajuće za početnike (slika 2). Posebnim prilagođenim dodatkom prevodi se Alice3 sintaktičko stablo u Java kod. Pri prenošenju projekta iz Alice3 u Java okruženje kontekst učenikovog projekta se ne mijenja (slika 3, slika 4). On se i dalje bavi svojim animiranim projektom kojeg je stvorio u Alice, ali sada kod svoj projekta može mijenjati tradicionalnim uređivanjem teksta. Na taj način ostvarena je veza između strategije rješavanja problema u Alice okruženju s podrškom tehnike poučavanja **posrednog transfera** poznatog pod imenima **premošćivanje** (engl. **bridging**) i **grljenje** (engl. **hugging**). U tehnici premošćivanja učitelj pomaže učeniku stvoriti poveznicu iz sadržaja u kojem je koncept naučen do nekog drugog mogućeg sadržaja. Korištenjem tehnike grljenja učitelj stvara situacije učenja koje su veoma slične situacijama u kojima se očekuje transfer. Sličan pristup koristi se u BlueJ okruženju. To je obrazovni alat koji se koristi za početno poučavanje objektno orijentiranog programiranja, a nudi grafičku prezentaciju klasa, u obliku UML dijagrama. Korisnik može kreirati svoje klase te pozivati metode kojima objašnjava djelovanje tih klasa [50].

Ipak, mnoge programere početnike, pri pisanju prvog računalnog programa, prati osjećaj neuspjeha jer programiranje izaziva neočekivana ponašanja programa kao što su sintaktička pogreška, pogreška izvođenja programa ili neka izlazna vrijednost programa koju početnik nije očekivao. Sve navedeno može se ubrojiti u oblike povratnih informacija. Povratne informacije ključne su za pomaganje učenicima u razumijevanju samog programa te načina na koji računala interpretiraju program. Vještine učenika moraju se razvijati baveći se i neispravnim programima kojima su učenici potaknuti na razvoj slučajeva za testiranje. Navesti ćemo nekoliko načina uvođenja procesa testiranja u uvodne predmete programiranja koji su se pokazali uspješnima.

Problemsko orijentirano programiranje i testiranje POPT [17] metoda je poučavanja podržana od strane alata TestBoot koja omogućuje početnicima definiranje jednostavne tablice ulazno-izlaznih slučajeva bez potrebe učenja novog okruženja ili mijenjanja strukture programa. Takav pristup proizveo je bolju kvalitetu generiranog koda, manji broj ispravaka početnog programa te dulje vrijeme za stvaranje prvog rješenja. Navodi nas na razmišljanje da je ovaj pristup uspio stimulirati učenike na bolje razmišljanje o rješenjima koje implementiraju [17].

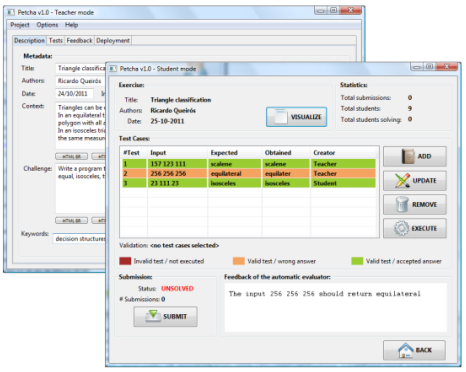
Drugi pristup uključivanja koncepata testiranja u uvodni predmet programiranja koristi neko personalizirano programskog okruženja kao što je npr. okruženje Gidget [51]. Gidget (slika 5) je igra u kojoj početnik pomaže robotu popraviti njegov neispravan kod. Početnici su vođeni kroz niz razina kojima uče dizajn i analizu osnovnih algoritama u jednostavnom imperativnom jeziku. Osnovni zadatak igre je naučiti komunicirati s robotom naredbama, a sve kako bi robot postigao određeni cilj. Prvih devet razina bave se učenjem osnovne sintakse robota. Gidget je prikazan kao čovjekoliki robot koji mijenja izraze lica ovisno o tome koliko je učenik bio uspješan u izvedbi programa. U slučaju pogreške u kodu robot preuzima odgovornost za pogrešku na sebe i ljubazno moli učenika za pomoć u ispravljanju pogreške (slika 6). Takav pristup izazvao je povećanje motivacije te angažmana učenika u korigiranju neispravnog koda [51].

Slika 5: Gidget- sučelje u kojem učenici pomažu robotu popraviti neispravne programe



Slika 6. Gidget -Pogreška izvođenja s odgovarajućom porukom robota

Kao primjer alata koji predstavlja pomoć u poučavanju učenika (engl. **teaching assistant – TA**) navedimo alat **Petcha**. Petcha je alat koji djeluje kao automatizirani demonstrator u predmetima programiranja. Krajnji cilj ovog alata je povećanje broja vježbi programiranja koje su učenici učinkovito riješili. Petcha to ispunjava pomažući nastavnicima, autoru vježbi programiranja i učenicima pri rješavanju zadataka. Alat također integrira u cjelinu automatski program za ocjenjivanje, učenje sustava za upravljanje, biblioteke objekata učenja te samo okruženje programiranja (slika 7). Ovaj alat može se smatrati i prijelaznim alatom u učenju programiranja jer nadopunjuje postojeće alate, a lako se može ukloniti u trenutku kada više nije potreban. Umjesto da pruža vlastita okruženja za rješavanje, ovaj alat promovira uporabu postojećih razvojnih okruženja kao što su Eclipse i Visual Studio. Primjenom alata Petcha u uvodnom predmetu programiranja povećao se broja uspješnih rješenja učenika [52] [53].



Slika 7. Sučelje alata PETCHA s učenikovim i učiteljevim Čvorovima, prema **[52]**

IV. PREDIKTORI IZVEDBE PROGRAMERA POČETNIKA

Najbolji indikatori uspjeha u svim disciplinama smatraju se samopredviđeni uspjeh, stav učenika, njegovo oduševljenje te opća akademska motivacija. Ovi faktori imaju snažan utjecaj na izvedbu učenika, ali ne razdvajaju programiranje od ostalih disciplina. Kako bismo pronašli specifične indikatore potrebno je provesti osjetljivija istraživanja u području izvedbe programiranja. Najčešće su promatrani faktori kao što su stručnost u svom govornom jeziku, broj programskih jezika koji se koriste i analiziraju, matematičke sposobnosti, glazbene sposobnosti, prethodni akademski stupanj, mjere opće inteligencije, samopouzdanje ... Ranija opsežnija istraživanja između 12 mogućih prediktora [54] kao najsnažnije prediktore ističu upravo stupanj ugode, matematičko predznanje te atribuciju uspjeha ovisnog o sreći. Fincher i suradnici [55] provode zanimljivo višenacionalno, višeinstitucijsko istraživanje uvodnih predmeta programiranja koje je pokazalo da je dubinski pristup učenju u pozitivnoj korelaciji s ocjenom iz početnog programiranja dok površinski pristup učenju ukazuje na negativnu korelaciju s istom ocjenom. Otkrivena je značajna pozitivna korelacija između ocjene na kraju predmeta i skiciranja prostornih mapa. Učenici koji su bili sposobni stvoriti prostornu mapu koja se sastoji od poznatih objekata i sljedova lokacija postizali su bolje rezultate u programiranju od onih koji su bili u stanju nacrtati samo smjerove kretanja. To ukazuje da različite navigacijske strategije mogu pozitivno utjecati na programerski kod i na taj način formirati konceptualizaciju prethodnog računalnog iskustva. Neka druga istraživanja proučavala su utjecaj spola, materinjeg jezika učenika, stila učenja te izvedbe učenika pri rješavanju problema u drugim disciplinama kao što su matematika i ostale prirodne znanosti. Byrne (2001) i Chumra (1998) ukazuju da spol nema utjecaja na izvedbu programiranja. Slično, Allert (2004) i Byrne (2001) pronalaze da prethodno računalno iskustvo također nije imalo utjecaja na izvedbu programiranja, no u skladu s rezultatima Byrne i Cumra postoji snažna pozitivna korelacija između izvedbe učenika u predmetu programiranja s rješavanjem problema u predmetima kao što su matematika i znanost. Pillay i Jugoo [56] također ukazuju da izvedba programiranja ne ovisi o spolu, da prethodno računalno iskustvo ne utječe na postizanje boljih rezultata kod učenika, no da sposobnost rješavanja problema te prvi programski jezik imaju značajan

pozitivan utjecaj na uspješnu izvedbu programiranja. Istraživanja učeničkih komentara i primjedbi prikupljenih za vrijeme zadataka programiranja pokazala su da pozitivna prethodna iskustva programiranja podržavaju pozitivnu samo-procjenu učenika u radu [57]. Uporaba ispravnih mentalnih modela koncepata programiranja pozivno utječe na uspješno početno programiranje [58].

Detaljnom usporedbom eksperata i programera početnika uočeni su neki „pozitivni“ i „negativni“ prediktori programiranja [59]. Dobrim prediktorom programiranja pokazalo se dobro skiciranje prostornih mapa [55, 59, 60], sposobnost indukcije (uočavanja pravila iz primjera), uporaba održivih mentalnih modela [9, 59, 58], smisao za humor, strpljenje i upornost, vještina manipulacije pravilima gramatike i sintakse. Negativni prediktori su oni faktori koji su pokazali negativnu korelaciju s uspjehom u izvedbi programiranja kao što su: veći broj sati proveden u igranju računalnih igara, česte frustracije s tendencijom brzog odustajanja, odbojnost prema programiranju, osjećaj neuspjeha pri programiranju, uporaba različitih programerskih modela za rješavanje problema, vjerovanje da je rješenje najbolje naučiti napamet da bi se naučilo, vjerovanje da pravi programeri odmah vide rješenje problema, vjerovanje da je potrebno veliko znanje o programiranju za postizanje uspjeha [59].

Veći broj do sada provedenih istraživanja temelji se na statističkim testovima, koji često nisu u mogućnosti dohvatiti promjenu učeničkog napretka u učenju. Za razliku od tradicionalnih prediktora u koje ubrajamo prethodno iskustvo programiranja [56], matematičko predznanje [54], osobine ponašanja, samopoštovanje, stilove učenja [56], strategije učenja [55], vještine prostorne vizualizacije [55], prvi programski jezik [56] te atribuciju uspjeha [57], Watson i Shrock definiraju 12 novih prediktora na temelju aspekata ponašanja učenika pri programiranju kako bi odrazili napredak u učenju koji se događa tijekom vremena. Takvi prediktori u stanju su dinamički identificirali učenike s poteškoćama u učenju, a mogu se i primijeniti za izradu nekog ekspertnog

Tablica 1: Pregledni prikaz autora koji su istraživali faktore značajne za izvedbu programiranja, rema metodologiji [2]

			Postoji korelacija		**Ne postoji korelacija (nije prediktor)**
			Pozitivna korelacija	**Negativna korelacija**	
Potencijalni prediktori uspjeha u programiranju	Matematičko predznanje		Wilson i Shrock, 2001 [54] Stein, 2002 White i Sivitanides, 2003		
	Skiciranje prostornih mapa		Fincher [55] Kranch [59] Watson i sur. [61]		
	Osjećaj ugode		Bergin i sur., 2005		
	Atribucija uspjeha		Wilson, Shrock, 2001 [54] Henry i sur., 1994 Ventura, 2005 Kinnunen i Simon, 2014 [57]		
	Stilovi učenja	prema Kolbu			Pillay, Jugoo [56]
		Prema Gregorcu	Watson i sur. [61]		
	Prvi programski jezik		Pillay, Jugoo [56]		
	Vještine prostorne vizualizacije		Fincher [55]		
	Ponašanje pri programiranju	visok postotak uzastopnih uspješnih prevođenja programa	Watson i sur. [61]		
		Visok postotak uzastopnih pogrešaka u vježbama programiranja		Watson i sur. [61]	
		veći broje sati utrošen na pogreške		Watson i sur. [61]	
	Strategije učenja	Dubinski pristup	Fincher [55]		
		Površinski pristup		Fincher [55]	
	Spol				Byrne, 2001 Chumra, 1998 Pillay, Jugoo [56]
	Prethodno programersko iskustvo				Allert, 2004 Byrne, 2001 Pillay, Jugoo [56]
	Broj godina programiranja				Watson i sur. [61]
	Broj upoznatih programskih jezika				Watson i sur. [61]
	Uporaba održivih mentalnih modela		Bornat i sur. [58] Kranch [59] Kranch [9]		
	Sposobnost rješavanja problema		Pillay, Jugoo [56]		
	Velik broj sati igranja računalnih igara			Kranch [59]	
	vježbanje jezičnih vještina		Siegmund i sur. [62]		

sustava koji bi odgovarajuće reagirao prema učenicima kad je to potrebno. Ovo istraživanje ukazuje da je ponašanje učenika pri programiranju jedan od najsnažnijih indikatora uspjeha [61]. Rezultati su otkrili snažnu pozitivnu povezanost s dobrim uspjehom u programiranju upravo za učenike koji su imali visok postotak uzastopnih uspješnih prevođenja programa. Učenici s lošijim uspjehom u programiranju imali su visok postotak uzastopnih pogrešaka u vježbama programiranja te veći broj sati utrošen na ispravljanje pogrešaka u programu. Iako se prethodno programersko iskustvo pokazalo korisnim za učenike, broj godina programiranja te broj programskih jezika s kojima je učenik upoznat nije pokazao značajan utjecaj na njihovu izvedbu [61]. Rezultati ukazuju na malu povezanost stilova učenja prema Kolbu s uspjehom u programiranju, što je u skladu s prethodnim istraživanjima [56], ali i značajnu pozitivnu povezanost stilova učenja prema Gregorcu, za dimenziju apstraktan/slučajan (engl. abstract/random), zbog čega stilove učenja prema Gregorcu možemo smatrati razumnim indikatorom uspjeha u programiranju. Pregledni prikaz autora koji su se bavili istraživanjem faktora koji mogu biti značajni za izvedbu programiranja prikazan je u Tablici 1.

Većina promatranih faktora dio su osobnih epistemiologija učenika računarske znanosti. Učitelj teško može bitno utjecati na većinu ovih faktora, ali je izuzetno važno da dobro poznaje svoje učenike. Tako može bolje udovoljiti potrebama učenika i olakšati im usvajanje zahtjevnih koncepata programiranja. Učitelj može prilagoditi strategije poučavanja, odabrati odgovarajući sustav za poučavanje te primjeren početni programski jezik. Novija istraživanja sve više pažnje posvećuju faktoru motivacije i angažmanu učenika računalne znanosti pa se istražuju razni oblici učeničkih poticaja. Nagrada na natjecanju odnosno dodatni bodovi na ispitu zbog uspjeha na natjecanju programiranja pokazala je sposobnost motiviranja učenika u uvodnom predmetu programiranja [63, 64, 65]. Takve strategije su se pokazale izuzetno motivirajuće za učenike, a samo postojanje konkurencije učenici su ocijenili izuzetno pozitivno. Najveći izazov za nastavnike računarske znanosti leži upravo u radu sa šarolikom skupinom učenika koji često dolaze s različitim predznanjima. Nastavniku je vrlo teško pronaći odgovarajući stupanj težine nastavnog sadržaja za sve učenika. Ako je razina prezentacije preniska, nekima od najboljih učenika bit će dosadno i osjećat će se demotivirani za rad. Neki učitelji zagovaraju novu metodologiju rada koja zamjenjuje završni ispit s nizom aktivnosti u kontekstu kontinuirane provjere znanja te uporabu online natjecanja programiranja u kojem sudjeluju svi učenici. Pristup se pokazao izuzetno učinkovitim, stopa prolaznosti predmeta uvodnog programiranja gotovo se udvostručila [66]. Ipak, u nedostatku većeg broj istraživanja koja imaju za cilj otkriti utjecaj natjecanja na uspjeh učenika u programiranju ove rezultate treba promatrati okvirno.

U proučavanju karakteristika koje utječu na izvedbu programera, kroz prethodna dva desetljeća, provedeni su eksperimenti koji su se oslanjali na razne tehnike kao što su usporedbe uspješnosti rješavanja zadataka upitnicima, kvizovima i projektnima zadacima. Uspoređivani su uspjesi djelovanja pojedinih strategija početnim i završnim testovima. Tehnikom glasnog artikuliranja misli programera pri rješavanju zadataka, kroz ankete i intervjue, stavljen je naglasak na samoizvješćivanje. Takve tehnike imaju za cilj dohvatiti napredak u učenju koji se događa tijekom vremena. Istovremeno takav pristup veoma je zahtjevan jer istraživač mora poduzeti snažne mjere zaštite osobito u fazi prikupljanja i analize podataka, npr. triangulacija metoda, istraživača i sl., kako bi osigurao valjanost i pouzdanost.

Slika 8. Tijek istraživanja, prema **[62]**

Potpuno novim smjerom u istraživanju karakteristika i faktora značajnih za izvedbu programiranja zaputili su se istraživači koji su funkcionalnom magnetskom rezonancom promatrali aktivnosti koje se događaju u dijelovima ljudskog mozga pri programiranju odnosno razumijevanju programa (slika 8). Istraživanje je ukazalo da se za vrijeme programiranja događaju aktivnosti u pet različitih područja ljudskog mozga koja su povezana s radnom memorijom, pozornošću i obradom jezika [62]. Obrada jezika pokazala se veoma važnom za razumijevanje rada programa što po prvi puta empirijski dokazuje Dijkstra-inu tvrdnju [67] da je vježbanje jezičnih vještina, osim vještina rješavanja problema veoma važno za učinkovito učenje programiranja.

V. ZAKLJUČAK

U raspravi o poučavanju programera početnika navedene su neke strategije poučavanja koje su pokazale potencijal da olakšaju učenicima usvajanje zahtjevnih koncepata programiranja. Većina navedenih strategija smješta učenika u središte učenja što istovremeno postavlja učiteljima računarske znanosti nove izazove mijenjanja tradicionalnih navika poučavanja. Novija istraživanja ukazuju da postoji još uvijek značajan broj učitelja koji iskazuju određeni otpor prema neophodnim promjenama u procesu poučavanja, iako je postignut napredak u razumijevanju i očekivanjima vezanim uz izvedbe učenika u programiranju [13]. Navedene strategije zahtijevaju od učenika veći angažman u radu i posebnu pažnju posvećuju svakom obliku međusobne interakcije i suradničkog angažmana. Niti jedna od navedenih strategija nije se pokazala

izuzetno uspješnom u svakoj situaciji učenja što navodi na zaključak da je potrebno prilagoditi odabir strategije kontekstu učenja, okruženju učenja te samom učeniku.

U zadnje vrijeme posebno su se pokazala zanimljiva istraživanja koja se bave pronalaskom prediktora dobrog uspjeha izvedbe programiranja. Iako na tom području postoji veći broj istraživanja, manji je broj faktora koji su se pokazali dobrim prediktorima u većem broju istraživanja pa su potrebna daljnja istraživanja i nad nekim novim faktorima kao što su npr. upotreba alata temeljenih na igrama, povezanost vizualizacije i nastavnog procesa programiranja ili upotreba alata za unapređenje načina prikazivanja pogrešaka početnicima. Istraživanja vezana uz stilove učenja pokazala su se zanimljivim, ali nedosljednim. Rezultati uglavnom ukazuju na slabe ili nikakve povezanosti u jednom istraživanju, a nešto snažnije povezanosti u drugom, pa se preporuča provođenje meta-studije koja bi uskladila metodologiju provedenih istraživanja te odredila neki valjan zaključak.

Navedena istraživanja faktora i strategija poučavanja značajnih za unapređenje znanja programera početnika ostavljaju prostora za daljnji rad. Zanimljivo bi bilo detaljnije istražiti utjecaj izbora prvog programskog jezika kao prediktora izvedbe programiranja. Rezultati istraživanja koji ističu pozitivnu povezanost jezičnih sposobnosti i izvedbe programiranja otvaraju pitanje početka učenja programiranja u što ranijoj dobi učenika primjenom neke od strategija poučavanja namijenjene početnicima npr. strategije programiranja u paru ili strategije izravnog uključenja.

BIBLIOGRAFSKI NAVOD

[1] Sheard, Judy; Carbone, Angela, »ICT teaching and learning in a new educational paradigm: Lecturers' perceptions versus students' experiences,« u *Proc. of the Seventh Baltic Sea Conference on Computing Education Research*, Koli National Park, Finland, 2007.

[2] Webster, Jane; Watson, Richard T., »ANALYZING THE PAST TO PREPARE FOR THE FUTURE: WRITING A LITERATURE REVIEW,« *MIS Quarterly, Webster & Watson/Guest Editorial*, svez. Vol. 26, br. No. 2, June, 2002.

[3] Kafai, Yasmin B.; Burke, Quinn, »The Social Turn in K-12 Programming: Moving from Computational Thinking to Computational Participation,« u *SIGCSE'13, March 6–9*, Denver, Colorado, USA, 2013.

[4] Kollock, Peter;Smith, Marc, »Communities in Cyberspace,« *Communities in Cyberspace, London: Routledge.*, p. 3–25, 1999.

[5] Ben-Ari, Mordechai, »Constructivism in Computer Science Education,« u *SIGSCE 98*, Atlanta, GA, USA, 1998.

[6] Fincher, Sally, »What are We Doing When We Teach Programming?,« u *29th ASEE/IEEE Frontiers in Education Conference*, San Juan, Puerto Rico, 1999.

[7] Piteira, Martinha;Costa, Carlos, »Computer Programming and Novice Programmers,« u *ISDOC'12*, Lisbon, Portugal, June 11, 2012.

[8] T. T. Yuen, »Novices' Knowledge Construction of Difficult Concepts in CS1,« *SIGCSE Bulletin* , pp. 49-53, 4 November 2007.

[9] Kranch, D. A., »Teaching the novice programmer: A study of instructional sequences and percetion,« *Educ Inf Technol*, pp. 291.-313., 2012, vol:17.

[10] Kolikant, Y. B-D., »Students' alternative standards for correctness,« u *ICER '05:Proceedings of the 2005 international workshop on Computing education research*, Seattle, WA, USA, 2005.

[11] Ma, Linxiao, Investigating and Improving Novice Programmers' Mental Models of Programming Concepts, University of Strathclyde, Department of Computer & Information Science, 2007.

[12] McCracken, M.; Almstrum, V.; Diaz, D.; Guzdial, M.; Hagan, D.; Kolikant, Y. B.; Laxer, C.; Thomas, L.; Utting, I; Wilusz, T, »A multi-national, multi-institutional study of assessment of programming skills of first-year CS students,« u *SIGCSE Bull, 33, 4*, 2001.

[13] Utting, Ian; Bouvier, Dennis; Caspersen, Michael, »A Fresh Look at Novice Programmers' Performance and Their Teachers' Expectations,« u *ITiCSE-WGR'13*, Canterbury, England, UK, 2013,.

[14] Biggs, J.B., »Teaching for quality learning at university,« u *Open University Press/Society for Research into Higher Education, second edition*, Buckingham, 2003.

[15] Raadt, Michael de; Hamilton, Margaret; Lister, Raymond; Tutty, Jodi; Baker, Bob; Box, Ilona; Cutts, Quintin; Fincher, Sally; Hamer, John; Haden, Patricia; Petre, Marian; Robins, Anthony; Simon; Sutton, Ken; Tolhurst, Denise, »Approaches to learning in computer programming students and their effect on success,« u *28th HERDSA Annual Conference*, Sydney, Australia, 2005.

[16] Buffardi, Kevin; Edwards, Stephen H., »Effective and Ineffective Software Testing Behaviors by Novice Programmers,« u *ICER'13, August 12–14*, San Diego, California, USA, 2013.

[17] Vicente Lustosa Neto 1, Roberta Coelho; Leite, Larissa; Guerrero, Dalton S.; Mendonça, Andrea P., »POPT: A Problem-Oriented Programming and Testing Approach for Novice Students,« u *ICSE 2013:Software Engineering in Education*, San Francisco, CA, USA, 2013.

[18] Vihavainen, A.; Paksula, M.; Luuklainen, P., »Extreme Apprenticeship method in Teaching Programming for Beginners,« u *Proceedings of the 42nd ACM technical symposium on Computer science*, 2011.

[19] Black, Toni R., »Helping Novice Programming Students Succeed,« u *JCSC 22, 2*, 2006.

[20] Vihavainen, Arto; Vikberg, Thomas; Luukkainen, Matti; Kurhila, Jaakko, »Massive Increase in Eager TAs: Experiences from Extreme Apprenticeship-based CS1,« *ITICSE 2013*, pp. 123-128, 1-3 july 2013.

[21] Phuong, Dinh Thi Dong; Shimakawa, Hiromitsu, »Collaborative cLearning Environment to Improve Novice Programmer with

Convincing Opinions,« *WSEAS TRANSACTIONS on Advances in Engineering Education, Issue 9, volume 5,* pp. 635-644, September 2008.

[22] Corney, Malcolm; Teague, Donna; Thomas, Richard N., »Engaging Students in Programming,« u *12th Australian Computing Education Conference (ACE2010)*, Brisbane, Australia, 2010.

[23] Porter, Leo; Garcia, Saturnino; Glick, John; Matusiewicz, Andrew; Taylor, Cynthia, »Peer Instruction in Computer Science at,« *ITiCSE'13,,* 1-3 July 2013.

[24] Raadt, Michael de; Toleman, Mark; Watson, Richard, »Training Strategic Problem Solvers,« *SIGSCE,* pp. volume 36, number 2 , pages: 48-51, June 2004.

[25] Hu, Minjie; Winikof, Michael; Cranefield, Stephen, »Teaching Novice Programming Using Goals and Plans in a Visual Notation,« u *Proceedings of the Fourteenth Australasian Computing Education Conference (ACE2012)*, Melbourne, Australia, 2012.

[26] Porter, Ron; Calder, Paul, »A Pattern-Based Problem Solving Process for Novice Programmers,« u *ACE 2003, vol. 20*, Adelaide, Australia, 2003.

[27] Solloway, Elliot, »Learning to Program = Learning to Construct Mechanisms and Explanations,« *Communications of the ACM, Volume 29, Number 9*, pp. 850-858, September 1986.

[28] Ma, L.; Ferguson, J.; Roper, M.; Wood, M., »Investigating the viability of mental models held by novice programmers,« *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education,* svez. 1, br. ISBN: 1-59593-361-1, pp. 499-503 , 2007.

[29] Lathinen, Essi; Ahoniemi, Tuukka, »Kick-Start Activation to Novice Programming - A Visualization - Based Approach,« *Electronic Notes in Theoretical Computer Science*, br. 224, pp. 125-132, 2009.

[30] Naps, Thomas; Cooper, Steven; Koldehofe, Boris; Leska, Charles; Rößling, Guido; Dann, Wanda; Korhonen, Ari; Malmi, Lauri; Rantakokko, Jarmo; Ross, Rockford J.; Anderson, Jay; Fleischer, Rudolf; Kuittinen, Marja; McNally, Myles, »Evaluating the Educational Impact of Visualization,« u *ITiCSE-WGR '03: Working group reports from ITiCSE on Innovation and technology in computer science education*, 2003, December.

[31] Ma, Linxiao; Ferguson, John; Roper, Marc; Ross, Isla; Wood, Murray, »Improving the Mental Models Held by Novice Programmers Using Cognitive Conflict and Jeliot Visualisations,« u *ITiCSE'09*, Paris, France, 2009.

[32] Hundhausen, Christopher D.; Douglas, Sarah A.; Stasko, John T., »A Meta-Study of Algorithm Visualization Efectiveness,« *Journal of Visual Languages and Computing 13,* pp. 259-290, 2002.

[33] Pauch, R.; Kelleher, C., »Lowering the barriers to programming: A taxonomy of programming environments and languages for novice,« *ACM Computing Survey 37, 2,* June 2005.

[34] Greenberg, Ira; Kumar, Deepak; Xu, Dianna, »Creative Coding and Visual Portfolios for CS1,« u *SIGCSE 2012*, raleigh, NS, USA, 2012.

[35] Raadt, Michael de, »Introductory Programming in a Web Context,« *12th Australian Computing Educational Conference (ACE), vol. 103,* 2010.

[36] McGill, M. M., »Learning to Program with Personal Robots: Influences on Student Motivation,« *ACM Transactions on Computing Education,Vol. 12, No. 1,* p. 32 pages, March 2012.

[37] Fagin, Barry; Merkle, Laurence, »Measuring the Effectiveness of Robots in Teaching Computer Science,« u *SIGCSE'03*, Reno, Nevada, USA, 2003.

[38] Summet, Jay; Kumar, Deepak; O'Hara, Keith; Walker, Daniel; Ni, Lijun; Blank, Doug; Balch, Tucker, »Personalizing CS1 with Robots,« u *SIGCSE'09*, Chattanooga, Tennessee, USA, 2009.

[39] Guzdial, M., »A media computation course for non-majors,« u *Proceedings of the 8th annual conference on Innovation and technology in computer science education*, Thessaloniki, Greece, 2003.

[40] Powers, Kris; Ecott, Stacey; Hirshfield, Leanne M., »Through the Looking Glass: Teaching CS0 with Alice,« u *SIGCSE '07*, Convington, Kentucky, USA, 2007.

[41] Dann, Wanda; Cosgrove, Dennis; Slater, Don; Culyba, Dave; Cooper, Steve, »Mediated Transfer: Alice 3 to Java,« u *SIGCSE'12*, Raleigh, North Carolina, USA, 2012, March.

[42] Adams, Joel C.; Webster, Andrew R., »What Do Students Learn About Programming From Game, Music Video, And Storytelling Projects?,« u *SIGCSE'12*, Raleigh, North Carolina, USA, 2012.

[43] Stolee, Kathryn T.; Fristoe, Teale , »Expressing Computer Science Concepts Through Kodu Game Lab,« u *SIGCSE'11*, Dallas, Texas, USA, 2011, March 9–12.

[44] »Ministry of Truth,« 10th November 2008. [Mrežno]. Available: http://theministryoftruth.tumblr.com/post/58975997/the-encyclopedia-of-computer-languages. [Pokušaj pristupa 2 may 2014].

[45] McIver, L., »The Effect of Programming Language on Error ates of Novice Programmers,« u *12th Workshop of the Psychology of Programming Interest Group*, Cozenza Italy, 2000.

[46] Gupta, Diwaker, »What is a Good First Programming Langugage?,« *Crossroads - The ACM Student Magazine, Volume 10 Issue 4,* August 2004.

[47] Mannila, Linda; Raadt, Michael de, »An Objective Comparison of Languages for Teaching Introductory Programming,« u *Proceedings, Koli Calling*, 2006.

[48] Pillay, Nelishia, »A Study of Object-Oriented Design Errors Made by Novice Programmers,« u *SACLA '09*, Mpekweni Beach Resort, South Africa, 2009.

[49] Stefik, Andreas; Siebert, Susanna, »An Empirical Investigation into Programming Language Syntax,« u *ACM Transactions on Computing Education*, New York, NY 10121-0701 USA, 2013, November.

[50] Bennedsen, Jens; Schulte, Carsten, »BlueJ Visual Debugger for Learning the Execution of Object-Oriented Programs?,« *ACM Transactions on Computing Education, Vol. 10, No. 2, Article 8,* June 2010.

[51] Lee, Michael J.; Ko, Andrew J., »Personifying Programming Tool Feedback Improves Novice Programmers' Learning,« u *ICER'11*, Providence, RI, USA, 2011.

[52] Queirós, Ricardo; Leal, José Paulo, »PETCHA - A Programming Exercises Teaching Assistant,« u *ITiCSE'12*, Haifa, Israel., July 3–5, 2012.

[53] Queir'os, Ricardo Alexandre Peixoto de, A framework for practice-based learning applied to computer programming, Porto, Portugal: Departamento de Ciˆencia de Computadores, Faculdade de Ciˆencias da Universidade do Porto, 2012.

[54] Wilson,Brenda Cantwell; Shrock, Sharon, »Contributing to Success in an Introductory Computer Science Course: A Study of Twelve Factors,« u *SIGCSE 2001*, Charlotte, NC, USA, 2001.

[55] Simon; Fincher, S.; Robins, A.; Baker, Bob; Box, Ilona; Cutts, Quintin; Raadt, Michael de; Haden, Patricia; Hamer, John; Hamilton, Margaret; Lister, Raymond; Petre, Marian; Sutton, Ken; Tolhurst, Denise; Tutty, Jodi, »Predictors of Success in a First Programming Course,« *Australian Computing Education Conference (ACE)*, svez. 52, pp. 189-196, 2006.

[56] Pillay, Nelishia; Jugoo, Vikash R., »An Investigation into Student Characteristics Affecting Novice Programming Performance,« *inroads – The SIGCSE Bulletin, Volume 37, Number 4*, pp. 107-110, December 2005.

[57] Kinnunen, Paivi; Simon, Beth, »Ma program is OK - am I? Computing freshman's experience of doing programming assignements,« *Computer Science education*, svez. vol. 22, br. No. 1, pp. 1-28, March, 2014.

[58] Bornat, Richard; Dehnadi, Saeed; Simon, »Mental models, Consistency and Programming Aptitude,« u *ACE2008*, Wollongong, Australia, 2008.

[59] D. A. Kranch, Pisac, *Teaching STEM to Novices: Maximize Your Effectiveness and Minimize Your Losses.* [Performance]. 2011 STEMtech conference, 2011.

[60] Simon; Cutts, Quintin; Fincher, Sally; Haden, Patricia; Robins, Anthony; Sutton, Ken; Baker, Bob; Box, Ilona; Raadt, Michael de; Hamer, John; Hamilton, Margaret; Lister, Raymond; Petre, Marian; Tolhurts, Denise; Tutty, Jodi, »The Ability to Articulate Strategy as a Predictor of programming Skill,« u *ACE2006*, Hobart, Tasmania, Australia, 2006.

[61] Watson, Christopher; Li, Frederick W. B.; Godwin, Jamie L., »No Tests Required: Comparing Tradicional and Dynamic Predictors of Programming Success,« u *Sigcse 2014*, Atlanta, GA, USA, 2014.

[62] Siegmund, Janet; Kästner, Christian; Apel, Sven; Parnin, Chris; Bethmann, Anja; Leich, Thomas; Saake, Gunter; Brechmann, André, »Understanding Understanding Source Code with Functional Magnetic Resonance Imaging,« u *ICSE '14, May 31 – June 7*, Hyderabad, India, 2014.

[63] Khera, Vivek; Astrachan, Owen; Kotz, David, »The internet programming contest: a report and philosophy,« 1993, March.

[64] Widmer, Connie C.; Parker, Janet, »Programming contests: what can they tell us?,« *Journal of Research on Computing in Education, Volume 20, Issue 3*, pp. 287-295, 1988.

[65] Roberts, Eric, »Strategies for Encouraging Individual Achievement in introductory Computer Science Courses,« Austin TX, USA, March, 2000.

[66] Garcia-Mateos, Gines; Fernandez-Aleman, Jose Luis, »Make Learning Fun with Programming Contests,« *Transactions on Edutainment II, LNCS 5660, Berlin Heidelberg*, p. 246–257, 2009.

[67] Dijkstra ,Edsger W., »How do we tell truths that might hurt?,« *Selected Writings on Computing: A Personal Perspective, Springer-Verlag*, p. 129–131, 1982..