UNIVERZA V MARIBORU

FAKULTETA ZA NARAVOSLOVJE IN MATEMATIKO

Oddelek za matematiko in računalništvo

DIPLOMSKO DELO

Gregor Nemec

Maribor, 2015

KAZALO

PREGLED UPORABLJENIH SIMBOLOV IN OZNAČB

1. **IKT** - informacijsko komunikacijska tehnologija

1 UVOD

V svetovnem merilu se pojavlja trend po popularizaciji programiranja ali kodiranja. V zadnjem obdobju so se na spletu pojavili številni portali, kot je na primer *CodeAcademy*, ki ponujajo učenje programiranja.

Testiranje todojev

Testiranje novega ukaza!

V večini se učenci prvič srečajo s pojmi programiranja pri izbirnih predmetih *Urejanje besedil, Multimedija in Računalniška omrežja.* Dijaki se srečajo s programiranjem v 1. letniku pri predmetu informatike. Posebnost so strokovni programi, katerim je osnova računalništvo. Zanimali nas bodo novinci in njihove težave pri začetnih korakih učenju programiranja. Torej vsi učenci in dijaki, ki se šele srečujejo s programiranjem.

Najprej se je uporaba spletne tehnologija in nastanek spletnih portalov za namen učenja programiranja pojavila v akademskem okolju na posameznih univerzah. Zanimal nas bo razlog za nastanek takšnih okolij na univerzah, zato bomo pregledali literaturo in poskušali ugotoviti, zakaj in kako se na višje šolskem področju uporabljajo spletne tehnologije za poučevanje programiranja.

Izluščili bomo predlagane rešitve za uporabo spletnih tehnologij pri učenje programiranja. Spoznali bomo kaj so osnovni koncepti programiranja s katerimi se srečajo novinci in katere so strategije in metode, ki se pri učenju uporabljajo.

Na podlagi pregledanega bomo določili kriterije in kategorizirali ter ovrednotili spletne portale. Najbolj bojo zanimivi tisti spletni portali, ki ponujajo številne programske jezike, urejevalnik besedil, zaganjanje napisane programske kode in neko obliko odziva, ki uporabniku omogoča odkrivanje napak.

Ogledali si bomo kje se uči programiranja na osnovni (**OŠ**)in srednji šoli (**SŠ**). Pri katerih izbirnih vsebinah, predmetih in kakšna je vsebina, ki jo predvideva učni nart. Uporabo spletnih portalov bomo skušali umestiti v pouk OŠ in SŠ tako, da bo njihova uporaba najbolj koristna in smiselna.

2 UPORABA RAČUNALNIKA V IZOBRAŽEVANJU

Model uporabe računalnika v izobraževanju je Gerlič [2] razdelil na tri področja.

Primarno področje lahko uvrstimo učenje programiranja, saj sem prištevamo aktivnosti s katerimi želimo uporabnike seznaniti z delovanjem in uporabo računalnika oz. sodobno informacijsko komunikacijsko tehnologijo (**IKT**) [1]. Računalnik je tista učna vsebina, ki jo obravnavamo.

Sekundarno področje sem spada vse tiste aktivnosti, katere so vezane neposredno na izobraževalni proces katerega koli predmetnega področja. Računalnik in **IKT** nastopata kot učno sredstvo ali pripomoček v oblikah tradicionalnih računalniško podprtih učnih sistemov ali inteligentnih ekspertnih sistemov.

Terciarno področje, spadajo vse aktivnosti, ki spremljajo izobraževanje. Sem se štejejo aktivnosti izobraževanja, vodenja in upravljanja izobraževalnega sistema.

V tem diplomskem delu nas bo zanimalo le **primarno področje** uporabe računalnika v izobraževanju, ki je razdeljeno v dveh pomembnih področjih [2]:

- kot element splošne izobrazbe,
- kot element ožje strokovne - poklicne izobrazbe oz. usposabljanja.

2.1 SPLOŠNOIZOBRAŽEVALNO PODROČJE

V današnjem času se računalnik kot element splošne izobrazbe kaže kot velika potreba oz. se zdi znanje njegove uporabe samoumevno. Že pri najmlajših otrocih računalnik vzbuja zanimanje in interes. Računalnik je postal intelektualno orodje in pripomoček v vsaki sferi človekove dejavnosti in je prodrl tudi v šolo. Tako imenovana **računalniška pismenost** postaja nuja in zajema vse to kar bi človek moral znati o računalniku in to, kako je potrebno z njim delati, da bo uspešno živel v družbi, ki je osnovana na informacijah oz. informacijski družbi [3].

V zvezi z definicijo in pojmovanjem **računalniške pismenosti** se kažeta dve usmeritvi, Gerlič [2] navaja številne avtorje obeh usmeritev:

Prva poudarja **sposobnost računalniškega programiranj**a in opredeljuje s pojmom pismenosti sposobnost branja in pisanja podobno kot je to značilno za jezikovno pismenost. Tako zagovorniki, te smeri poudarjajo, da je cilj računalniške pismenosti, učenje in veščina programiranja z novim načinom mišljenja in strategijami ugotavljanja in popravljanja računalniških

programov.

Druga smer poudarja **splošno usposobljenost** za delo z računalnikom in da ni smiselno, da vsak kdo postane programer, zaradi tega, ker se bo računalnik uporabljal v najširšem smislu v praksi. Pomembno za učenca je m da razume, delovanje računalnika in se zaveda njegovega vpliva na razvoj družbe. Učencu moramo pomagati, da dejanske probleme identificira in jih lahko reši že z narejeno komercialno programsko opremo.

V zgodnjih letih sta bili značilni obe usmeritvi. Pozneje je prišlo do preobrata leta 1987 po mednarodnem simpoziju na Univerzi v Stanfordu. Eden od sklepov simpozija je bil ta, da se v splošnoizobraževalne programe, ne uči več programiranja, še posebej ne strukturiranih verzij programskega je na primer **BASIC**, temveč naj se uči uslužnostne programske opreme, kot je urejevalnik besedil, orodja za delo z podatkovnimi bazam, grafična grafična orodja itd..

Ta dejstva je Gerlič [2] povzel leta 2000 in je predlagal isto usmeritev na: *"Učencem vseh stopenj želimo ob čim večjem številu ur praktičnega dela z računalnikom, ob določenem problemu in ob uporabi ustreznih komercialnih programov seznaniti z osnovami računalništva in informatike."* V nadaljevanju bomo lahko ugotovili kakšni so današnji učni načrti za **OŠ** in **SŠ** in ali so se zgornje ugotovitve uresničile in ohranile.

Zanimivo bi bilo preiskati tudi kakšni so trendi danes? Ali zaradi boljše računalniške pismenosti, predvsem mlajših generacij obstaja potreba in trend, da se programiranje ponudi v širšem kontekstu tudi na splošnem izobraževalnem nivoju. Zanima nas koliko je računalniške znanosti in programiranja v splošnoizobraževalnem področju, saj želimo pokazati, da spletni portali za učenje programiranja zaradi tehnološkega napredka omogočajo lažjo pot k učenju programiranja. Zato nas po pozneje zanimalo kje je umeščeno v učnem načrtu programiranje v **OŠ** in **SŠ**.

2.2 Strokovno izobraževalno področje

2.3 Programiranje v OŠ

2.4 Programiranje v SŠ

3 Zgodovina programskih jezikov v izobraževanju

Uporaba računalništva v izobraževanju je bila deležna številnih sprememb. Sama uporaba računalnika v izobraževanju je tesno povezana z razvojem računalnikov. Začetno obdobje, 1960 letih prejšnjega stoletja so računalniki bili zelo dragi in veliki glavni računalniki (*ang.*

mainframe), na njih se je učilo programiranja, a so se uporabljali tudi za druga področja. //-> Terminalska obdobje, Poglej gerliča. V tem obdobju se je za učenje programiranja uporabljal **FORTRAN** ali **asembler**. Programi so bili majhi in enostavi, zaradi fizičnih omejiteh takratnega delovnega pomnilnika.

V 1970 so na trg prišli manjši računalniki, ki so bili tudi cenejši in zmogljivejši. V tem času pride v ospredje strukturirano programiranje. Najpopularnejši programski jezik je bil **PASCAL**.

V 1980 so se prvič pojavili samostojni osebni računalniki. Programski jeziki v tem obdobju so bili strukturirani in močnega tipa (*ang. strong type.*). Med te spada **Ada, Modul 2, ML in naj omenimo še Prolog**.

V naslednjem desetletji, 1990 so v ospredje prišli objektno orjentirani programski jeziki, kot sta **JAVA** in **C#** [8].

Metode poučevanja računalništva so se prav tako spreminjale. 1960 so računalnike uporabljali samo za poučevanje programiranja. Povdarek pri predmetih programiranja je bil predvsem na detaljlih zmožnosti programskega jezika. Programiranje je bilo omejeno le na reševanje enostavnih primerov in povdarek ni bil na reševanju problemov na splošno.

V 1970 je reševanje problemov in abstrakcija podatkov postala glavni in najpomembnejši del vseh programerskih predmetov, kar velja še danes. Programi so postali večji, bolj interaktivni in spremenil se je vnos podatkov z tekstovnega v grafičnega. Vsebina predmetov računalništva se je hitro razširjala, kakor so se množili številni programski jeziki [8].

4 UČENJE PROGRAMIRANJA

4.1 KAJ JE RAČUNALNIŠKA ZNANOST?

Računalniška znanost ima številne različice definicij, v grobem jo lahko definicijo strnemo v naslednjih trditvah [7].

- Ukvarja z značilnostmi tisktega kar je izračunljivo.
- Je znanost, ki izhaja iz več področij in ime korenine v matematiki, znanosti in inženirstvu.
- ima mnoga podpodročja in je interdisiplinarna z biologijo, ekonomujo, medicino, zabavo.

- Ime računalništvo ali računalniška znanost nas lahko tudi zavede in jo zamenjamo z področjem uporabe računalnika.

4.2 Kaj je programiranje?

4.3 Programske paradigme

Paradigma je način kako obravnavamo in gledamo na stavri, je okvir v katerem leži naša interpretacija realnosti sveta. Paradigma najpogosteje pomeni vzorec delovanja v znanstvenem ali drugem raziskovanju. Izraz -programske paradigme- je več pomenka, ki povzema mentalne procese, strategije reševanja problemov, povezave med različnimi paradigmami, programske jezike, stil programiranja in še več (Wikipedia: Paradigma) [7].

Povemo lahko, da je programiranja, hevristična paradigma za algoritme ki rešujejo probleme. Programski jezik je način za izražanje programske paradigme.

Programske paradigme so hevristike, ki se uporabljajo za reševanje problemov. Programska paradigma analizira problem, čez specifičen pogled in na ta način formulira rešitev za dani problem, ki ga razdeli na manjše dele med katerimi definira razmerja.

Programske paradigme so na primer proceduralno, objektno orientirano, funkcijsko, logično in istočasno programiranje.

V nadaljevanju bomo spoznali značilnosti dveh programskih paradigem.

4.4 Proceduralno programiranje

4.5 Objektno orientirano programiranje

4.6 Osnovni koncepti programiranja

V naslednjem odstavku se bomo vprašali kako lahko formuliramo sintakso programskega jezika? In kaj je npr. definicija *kopice*.

V ta namen definiramo mehko idejo po avtorju Hazzan [7], ki je naslednja. Mehka ideja je koncept, ki mu ne moremo pripisati toge, niti formalne definicije. Mehke ideje ni niti možno opisati z točno določeno aplikacijo. Na tem mestu se postavlja vprašanje kako lahko defineramo nekaj kar se odvija po korakih.

Da odgovorimo na zgornji dve vprašanji, lahko povemo, da so pravila sintakse togi orisi pri pisanju programske kode in da so semantična pravila mehke ideje. Opozorimo še na to, da koncepti v računalniški znanosti niso le toga pravila ali samo mehke ideje, temeč skupek obojega. V spodnji tabeli 1 prikazuje primer spremenljivke.

Tabela 1: Prikaz dvojnih, togih in mehkih orisov idej na primeru spremenljivke [7].

	togi orisi	**mehki orisi**
ime spremenljivke	Pravilo sintakse.	Potreba po imenu spremenljivke. Katero ime spremenljivke je pomembno in zakaj ga je potrebno določiti.
vrednost spremenljivke	Pravila tipa spremenljivke. Rezervacija pomnilnika.	Spremenljivka ima eno vrednost, ki se lahko spreminja s časom.
dodelitev začetne vrednosti	Pravila sintakse.	Pomen dodelitve začetne vrednosti

5 SPLETNI PORTALI ZA UČENJE PROGRAMIRANJA

Spletne portale za učenje programiranja bomo predstavili tako, da bomo najprej pregledali kaj so bili razlogi, da so se pojavili na univerzah, ki poučujejo računalniško znanost.

5.1 PREGLED SPUP V AKADEMSKEM OKOLJU

Spletni portali za učenje programiranja so nastali na različnih univerzah po svetu. V nadaljevanju bomo pregledali nekaj takšnih primerov. Zanimalo nas bo zakaj so se odločili za izdelavo takih portalov in katere težave so skušali z tem premostiti.

Na odprti univerzi v Hong Kong-u (**OUHK**) ponujajo tri računalniške sklope različnih težavnosti, za dodiplomske programe. Imajo zelo veliko populacijo študentov, ki se učijo programiranja. Avtorji tega članka [5] ugotavljajo, da je proces učenja programiranja kompleksen in zahteva veliko vaje programiranja. oz pisanja kode, izkaže se, da praktični del igra poglavitno vlogo v učnem procesu.

Glavna težava s katero se srečujejo na **OUHK**, je ta, da se s številom študentov, ki se vpišejo v smeri računalništva povečuje. Povečanje študentov pomeni manj časa za mentorstvo za posameznega študenta. Ob težavah, na katere naletijo študentje pri učenju programiranja, morajo dlje časa čakati na pomoč mentorja, kar zavira in slabša učni proces. Težava se še stopnjuje če študenti niso deležni tradicionalnega vpisa na univerze, kjer bi lahko bili vsak dan v stiku s svojimi kolegi in mentorji, ampak so deležni izobraževanja na daljavo.

Da bi študentje, lahko normalno sledili pouku na daljavo, si morajo doma urediti delovno okolje, kjer lahko programirajo. Študenti, dobijo vso potrebno učno literaturo in tudi programsko opremo, ki predstavlja *prevajalnik* in *razvojno okolje*.. Izkaže se, da imajo številni težavo nastaviti in se spoznati z integriranim razvojnim okoljem (*Integrated Development Environment (**IDE**)*) [5].

Težave pri izobraževanju na daljavi, se pojavijo tudi v komunikaciji. Študent, ki se izobražuje od doma in naleti na neko težavo, ki je ne ve sam rešiti, nima dostopa do svojih kolegov, ali mentorjev. Do mentorjev dostopa lahko le preko telefonskih klicev ali elektronske pošte. Če pogledamo še s strani mentorjev, imajo ti težavo z spremljanjem napredka takega števila študentov.

Zgoraj navedene težave rešuje jo s spletni portalom za učenje programiranja, ki uporablja programski jezik **JAVA**.

5.1.1 Sistemska arhitektura in pregled delovanja na OUHK

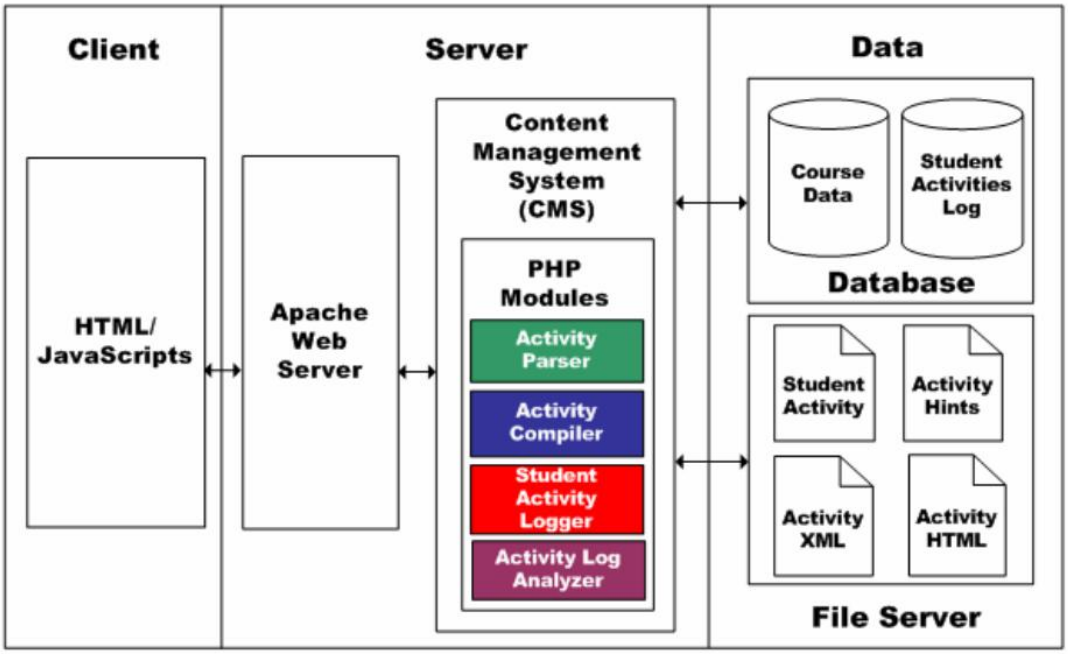
Zanimalo nas bo tudi kakšna je morebitna sistemska arhitektura takega spletnega portala, zato si bomo pomagali z primerom, arhitekture, ki so ga izdelali na **OUHK**. V nadaljevanju bomo govorili o *aktivnostih*, ki jih mora študent opraviti, to so naloge, programske rešitve na zastavljene probleme.

Kot prikazuje slika 1 je sistem urejanja vsebine (*ang. Contetnt Managment System (**CMS**)*), teče na spletnem strežniku Apache z MySQL podatkovno bazo. Sistem je narejen iz štirih pod modulov, ki so napisani v skriptnem jeziku PHP.

Ti moduli so naslednji, zajem aktivnosti, prevajalnik aktivnosti, dnevnik študentove aktivnosti in analizator dnevnikov aktivnosti. Samo delovanje je naslednje, ko odjemalec pošlje zahtevo za neko aktivnost, se ta naslovi strežniku, ki poišče programsko aktivnost. Z modulom *zajema aktivnosti*, strežnik zajame aktivnost, ki je zapisana v obliki **XML** in naloži vse potrebne datoteke. Zajem aktivnosti, prav tako naloži študentovo predhodno delo, ki je shranjeno datoteki aktivnosti. Ko se vse zajame in naloži se vsebina pošlje v obliki **HTML** nazaj k klientu.

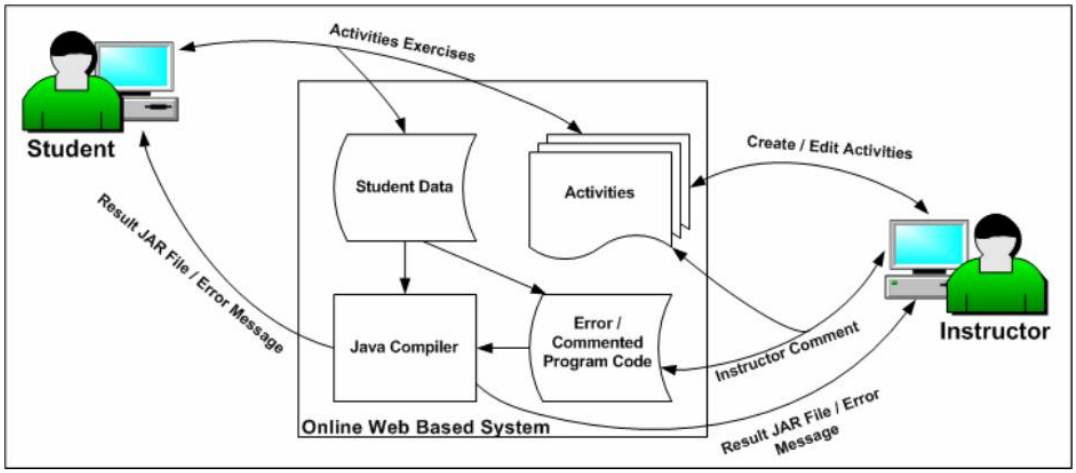
Strežnik omogoča tudi prevajanje aktivnosti. Ko strežnik dobi prošnjo za prevajanje programske kode, se ta prevede, če seveda v njej ni sintaktičnih napak in se ustvari datoteka **JAR**, ki jo študent lahko prenese s strežnika. Če so v programu napake, se ustvari dnevnik napake, v trenutni aktivnosti, prav tako se napaka izpiše na zaslonu študenta.

Vsako aktivnost zajame dnevnik študentove aktivnosti in jo shrani v podatkovno bazo. Z analizatorjem dnevnika študentove aktivnosti, mentorji dobijo vpogled v delo študenta in njegovega napredka.



Slika 1: Sistemska arhitektura spletnega portala za učenje programiranja, kot so jo naredili na OUHK [5].

V nadaljevanju opišimo, kako so si zamislili interakcijo med študentm in mentorjev s spletnim sistemom, diagram prikazuje slika 2. Spletni sistem omogoča študentom in mentorjem spletno okolje za učenje programiran. Mentorji na spletni portal naložijo snov preko spletnega brskalnika. Mentor lahko naloži datoteke opisom aktivnosti. Ta datoteka vsebuje osnovne opise in informacije o aktivnostih. Posebej naloži še datoteko v kateri je predloga za aktivnost. V to predlogo študent rešuje zadano nalogo. V posebno datoteko je naložen tudi namig, ta je študentu v pomoč in ponuja primer izpisa programa.



Slika 2: Prikaz med interakcijo študenta in mentorja s spletnim portalom [5].

Študent lahko pregleduje vse aktivnosti in si naloži katero koli izmed njih. Omogočeno ima, da program prevede na strežniku, ko prevajalnik naleti na napake strežnih vrne napako na

spletno stran. Če študent naleti na težavo, ki je povezana z reševanjem aktivnosti, lahko pošlje prošnjo za pomoč svojemu mentorju. Ko se mentor prijavi v sistem ima vpogled v napako in na začasno delovno datoteko študenta, mentor lahko zaganja prevajalnik na tem začasnem projektu študenta. Ko mentor popravi programsko napako, odgovori študentu in poda komentar na programsko kodo študenta. Študent ima vpogled v komentarje in predloge, ki jih je posredoval mentor.

Večina študentov smeri računalništva nima predhodnih izkušenj v programiranju z programskim jezikom **JAVA**. Sistem se uporablja kot spletno okolje za učenje programiranja. Študentom je s tem, dana množica aktivnosti oz. nalog, katere morajo sami uspešno opraviti. To lahko počnejo kadarkoli in od koderkoli. Študentom ni potrebno nastavljati programskega okolja, študenti vse programe, ki jih napišejo, lahko takoj prevedejo in jih zaganjajo na svojih računalnikih. Uporaba spletnega portala je pokazala da so študentje oddali 100% programskih nalog, napisanih v javi. To kaže na to, da so študentje samozavestno reševali naloge in jih oddajali. Pred uporabo spletnega portala je oddaja nalog bila 80%.

Kot pravijo avtorji članka in portala [5], je to šele začetek uporabe spletnega portala, ki nudi osnovno funkcionalnost. V nadaljevanju nameravajo dodati še inteligentni sistem, ki po nadzoroval napredek študentov.

5.2 Spletno okolje za izboljšavo mentalnih programerskih modelov

Naslednji članek, ki so ga sestavili avtorji z *Univerze Strathclyde iz Valike britanije*, se ukvarja z raziskovanjem vpliva nove strategije kognitivnega pristopa k poučevanju programiranja, ki spreminja mentalni model študentu tako, da v njem ustvari konflikt. V ta namen je bilo razvito tudi spletno okolje, ki implementira uporabo nove kognitivne strategije [6].

Kot pravijo avtorji v članku, z hitrim razvojem IKT, narašča tudi potreba po sposobnih programerjih in učenje programiranja postaja globalna skrb. V prvem letu pri predmetih programiranja, študenti obvladajo naloge programiranja dosti slabše kot bi to pričakovali. Slaba uspešnost s pozna predvsem na tem, da se mnogi izpišejo z smeri računalništva, takih je kar od 30% do 50%. Kot avtorji poudarjajo in povzemajo po drugih študijah so za to v glavnem krive težave pri reševanju problemskih nalog, ki nastopajo v programiranju. Nekatere druge študija vidijo krivca za neuspeh tudi v napačnem razumevanju ključnih konceptov pri programiranju, ki so lahko posledično krivi za težave pri reševanju problemov. Tradicionalni učni pristop je za učenje programiranja manj zanesljiv, da bi zagotovil pravilnost v razvoju mentalnih ali duševnih modelov o konceptih programiranja. Študije kažejo da študenti po enem letu predmeta programiranja še vedno nimajo pravih mentalnih modelov o osnovnih programskih konceptih.

Za izboljšanje mentalnih modelov avtorji predlagajo konstruktivno naravna učni model, ki vključuje strategijo kognitivnega konflikta in vizualiacijo programov. Zgodnje preizkušanje strategije kognitivnega konflikta pokažejo da so študenti bolj zavzeti za učni material in jih motivira tako, da si prej ustvarijo pravilno mentalno predstavo.

Z predlogom zgoraj omenjene strategije je nastalo spletno okolje, ki naj bi izboljšalo mentalne modele ključnih programskih konceptov. Učni model je sestavljen iz štirih korakov:

- **Predhodni korak:** Mentor razišče kakšni so predhodni mentalni modeli študentov in identificira neprimerne.
- **Korak kognitivnega konflikta:** V študentovi predstavi moramo sprožiti tak dogodek, ki v študentu izove neskladje z njegovo predhodno predstavo in s tem študenta potisnemo v konfliktno situacijo.
- **Korak konstruiranja modela:** Študentu s pomočjo visualizacije pomagamo ustvariti pravo mentalno predstavo.
- **Korak aplikacije:** Študent mora rešiti programsko nalogo z na novo ustvarjeno mentalno predstavo.

Spletno učno okolje podpira programski jezik **JAVA** in vodi dnevnik študentovega napredka. Za učenje programskih konceptov je na spletni strani vsak posamezen koncept povezan z potjo, ki predstavlja načrt potovanja. Poti koncepte se povezujejo tako, da se ti nadgrajujejo, saj znanje določenega koncepta potrebuje neko predznanje prejšnjega. Tako za razumevanje določevanja reference najprej potrebujemo predznanje o spremenljivkah ali npr. preden se študenti učijo kako s podajajo parametri v podprograme najprej morajo razumeti kaj je obseg nekega pod programa. Vrstni red spoznavanja programskih konceptov je pomemben. Med potmi so gumbi, ki predstavljajo vsak koncept. Na vsakem gumbu je označen rdeč križ kar pomeni, da študent še ni spoznal koncepta. Ko študent opravi naloge povezane s posameznim konceptom se rdeč križ spremeni v zeleno kljukico. Ko študent vstopi v koncept se izpiše študentova zgodovina z nalogami tega koncepta. Vsaka naloga vsebuje tako vprašanje, ki sproži konfliktno situacijo v mentalnem modelu študenta. Nato študenti dobijo učni material v visualni obliki. Za vizualizacijo uporabljajo orodje **Jeliot**, ki dinamično upodablja izvajanje javaskih programov. Za pravilnost razumevanje mentalnega modela mora študent odgovoriti na dodatna vprašanja. Če študentovi odgovori niso v skladu z podanim mentalnim modelom, dobi študent povratno informacijo o nepravilnem odgovoru. Naslednji korak je ta, da študent mora zagnati vizualizavijo dela programske kode, ki si ga je prej moral predstavljati. Tako ima možnost, da zazna nepravilnost v svojem mišljenju in tako lahko gradi na pravilnem konceptu.

Nekatere osnove težave, katere srečajo programerji novinci [8]:

1. Inštalacija in nastavitve okolja za programiranje.
2. Uporaba urejevalnika besedil.
3. Razumevanje napisanih nalog oz. problemov in uporabe sintakse programskega jezika pri pisanju programske kode.
4. Razumevanje napak prevajalnika.
5. Razhroščevanje.

V preteklosti je bilo razvitih mnogo orodij, ki so nastala ravno z raziskovanja učenja programiranja, vendar mnoga od teh zahtevajo, da študenti pišejo celotne programe od začetka do konca.

Tudi začetniki, ki uspešno premagajo začetne ovire in se lotijo takojšnjega programiranja, imajo zelo slabo napisano in konstruirano programsko kodo. Pomagati novincem, pistati kvalitetno programsko kodo je časovno zelo zahtevno opravilo.

Težave programiranja se stopnjujejo ko se za učenje programiranja uporabljajo Objektno-orjentirani programski jeziki, sej ti zahtevajo visoko stopnjo abstraktnega razumevanja programskih konceptov in so načrtovani predvsem za zahtevne programerje.

5.3 Problematika začetkov učenja programiranja

V začetku nas bo zanimalo kaj so spletni portali za učenje. Spoznali bomo, da poznamo različne kategorije spletnih portalov za posredovanje različnega znanja in veščin. Zanimali nas bodo predvsem spletni portali, ki učijo znanje programiranja.

Tradicionalni spletni portali v izobraževanju, kot so **moodle**, nikoli niso popolnoma izkoristili zmožnosti uporabe, ki jih ponujajo nove internetne in komunikacijske tehnologije. Večinoma so se uporabljale le kot podaljšana roka obstoječim metodam poučevanja. Uporabljale so se za objavo gradiv in spletno prijavo za oddajo nalog. Takšni sistemi ne zagotavljajo izboljšav kvalitete poučevanja programiranja [5].

Poglejmo primer spletnega portala, ki ga je izdelal avtor [8], in ima naslednje elemente.

1. Spletni portal za programiranja, ki omogoča naloge tipa “Zapolni prazna mesta”.
2. Ogrodje za analizo, ki preverja kvaliteto in pravilnost, nalog, tipa “Zapolni prazna mesta”.
3. Avtomatski sistem za dajanje povratnih informacij, ki sporoča prilagojena sporočila prevajalnika in formalni odziv študentom in njihovim mentorjem. Poročilo vsebuje kvaliteto napisanega programa, strukturo in pravilnost glede na programsko analizo.

5.4 Predlagane rešitve SPZUP na težave novincev

Pri samem vadenju programiranja je pomembno, da ob težavah, novinci dobijo čimprajšen odziv mentorja. V velikih razredih se to izkaže za zelo zahtevno. Z uporabo spletnih tehnologij so v pomoč prav spletni portali za učenje programiranja. Z njimi lahko razrešimo kar nekaj tegob, ki jih pestijo novince [8].

5.5 Prednosti spletnih portalov za učenje programiranja.

Ena od prednosti dela z takšnim sistemom je ta, da novinci niso odvisni od mentorjevih uradnih govorilnih ur, pravtako tako lahko naloge opravljajo kadar koli [8].

5.6 Primer sistemske arhitekture spletnega portala za učenje programiranja

Primer sistemske arhitekture kot so si zamislili avtorji [5]. Slika .. opis slike.

5.6.1 Analiza programske kode

Dober odziv spletnega portala mora dati poročilo o pravilnosti programa in o kvaliteti [8].

Ogrodje (ang. framework) za analizo programske kode naj bi vsebovalo:

- Sintaktično ali semantično opozarjanje na napake ali napake kompilerja. //To ima vgrajeno veliko spletnih mest.
- odziv na kvaliteto in pravilnost programske kode //Ali ga sistem nima ali je ta pomnanjlkjiv. //Zgornje pomaga predvsem slabpim učencem. //Večina sistemov izvaja statično analizo programske kode in tako ni v pomoč kakšne kvalitete je ta koda.
- Formalni odzin učitelja oz. komunikacija med učiteljem in učencem.

6 Strategije reševanja problemov

Programiranje je preces pri katerem rešujemo probleme. Reševanje problemov, zato mora biti središče poučevanja računalniške znanosti. Reševanje problemov je zahteven mentalni proces. Če na spletu pobrskamo za strategije reševanja problemov lahko hitro ugotovimo na

obstajajo različne strategije. Kot so recimo naštete na strani Wikipedia:Reševanje problemov (*ang. Problem solving*), abstrakcija, analogija, brainstorming, deli in vladaj in mnoge druge. Proces in tehnike reševanje problemov se uporablja v mnogih tehničnih in znanstvenih disciplinah [7].

V nekaterih primerih učenci sami razvijejo strategijo s katero rešijo nek problem. Otroci si na primer sami izmislijo enostvno seštevanje in odštevanje, dolgo pred tem kadar se to učijo pri pouku matematike. Toda brez formalne podpore za unčikovito strategijo reševanja problemov, spodleti še tako inovativnemu učencu tudi pri enostavnih strategijah kot je **preizkus in napaka**. Zato je pomembno, da se uči strategij za reševanje problemov.

6.1 Proces reševanja problemov

Vsak osnoven proces, ki se ukvarja z raševanjem problemov, ne glede na znanstveno disciplino, se začne z opisom problema. Vsak problem se navadno zaključi z neko rešitvijo, ki je v nekaterih primerih izražena z **zaporedjem korakov** ali **algoritmom**. V računalništve algoritem zapišemo z kodo nekega programskega jezika. Zapisan algoritem testiramo tako, da kodo zaženemo, jo izvedemo. Preden pridemo od opisa problema do podane rešitve moramo prehoditi kar nekaj težkih korakov. Na te vmesne korake lahko gledamo kot na procese odkrivanja, zato lahko na reševanje problemov gledamo tudi kot na kreativen, umetniški proces [7].

Splošno priznani koraki reševanja procesov so naslednji:

1. *Analiza problema.* Najprej je pomembno da razuemo kaj je problem in ga znamo identificirati. Če tega ne znamo, ne moremo priti do nobene rešitve.
2. *Alternativnie rešitve.* Razmišljamo o alternativnih rešitvah kako bi lahko rešili nek problem.
3. *Izbira pristopa.* Izberemo primeren pristop, kako rešiti problem.
4. *Razgradnja problema.* Problem razgradimo na manjše podprobleme.
5. *Razvoj algoritma.* Algoritem razvijamo po korakih, ki smo jih določili v podproblemih.
6. *Pravilnost algoritma.* Preverjanje pravilnosti algoritma.
7. *Učinkovitost algoritma.* Izračunamo učinkovitost algoritma.
8. *Refleksija.* Naredimo refleksijo in analizo na pot, ki smo jo naredili pri reševanju problema in naredimo zaključek z tem kar lahko izboljšamo za naslednji problem, ki ga bomo reševali.

Točen recept kako se lotiti reševanja ne obstaja. Učencem lahko le pokažemo nekatere metode in strategije, ki jim lahko pomagajo pri reševanju problemov. Poglejmo še nekatere pomembne korake podrobneje.

6.1.1 Razumevaje problema

Razumevanje problemov je prva stopnja v procesu reševanja problemov. Pri reševanju algoritemskih nalog najprej moramo prepoznati, kaj so vhodni podatki in kateri podatki naj bi bili izhodni. Če znamo povedati kaj bodo vhodni podatki, razumemo tudi bistvo samega problema.

6.1.2 Načrtovanje rešitve

Novinci se spopadajo z največjimi težavami na začetni stopnji načrtovanja rešitve za nek problem. V nadaljevanju so predstavljene tri strategije, ki jih lahko uporabimo na tem koraku reševanja problema.

Definicija spremenljivk problema: Pri rešitvi problema si pomagamo tako, da ugotovimo kaj morajo biti vhodni in kateri bojo izhodni podatki. S tem razjasnimo problem. V naslednjem koraku definiramo **spremenljivke**, ki so potrebne za rešitev problema.

Postopno izboljševanje (*ang. Stepwise Refinement)*: Po tej metodi nas najprej zanima celoten pregled strukture problema in odnosi med posameznimi deli. Zatem se šele poglobimo specifični in kompleksni implementaciji posameznih pod problemov. Postopno izboljševanje je metodologija, ki poteka od **zgoraj-navzdol**, torej od splošnega k specifičnemu. Drugačen pristop je od **spodaj-navzgor**. Za oba pristopa velja da eden drugega dopolnjujeta. V obeh primerih je problem razdeljen na manjše pod probleme ali naloge. Glavna razlika med obema je mentalni proces, ki je potreben za en ali drugi pristop. V nadaljevanju se posvetimo samo pristopu od **zgoraj-navzdol**. Rešitev, ki jo poda **postopno izboljševanje** ima modularno obliko, ki jo:

1. jo lažje razvijamo in preverjamo,
2. jo lažje beremo in
3. nam omogoča, da uporabljamo posamezne pod rešitve tudi za reševanje drugih problemov.

Algoritemski vzorci: Algoritemski vzorci združujejo matematični pogled in elemente načrtovanja. Vzorec podaja načrt na rešitev, s katero se srečamo mnogokrat. Algoritemski vzorci so primeri elegantnih in učinkovitih rešitev problemov in predstavljajo abstraktni

model algoritemskega procesa, katerega lahko prilagodimo in ga integriramo v rešitve drugim problemom.

Pri tem procesu lahko nastopi težava prepoznave vzorca algoritma pri novincih, saj ti niso sposobni prepoznati podobnosti med posameznimi algoritmi ali ne znajo prepoznati bistvo problema, njihove posamezne komponente in razmerja med njimi, da bi lahko rešili nove probleme. V takih primerih novinci radi ponovno izumijo že njim poznane rešitve, ki bi jih lahko uporabili. Te težave navadno nastanejo zaradi slabe organizacije sistematike znanja o algoritmih.

Proces reševanja problemov z algoritemskim vzorcem se navadno začne z prepoznavanjem komponent, ki vodijo k rešitvi in iskanjem podobnih problemov, na katere še imamo znane rešitve. Zatem prilagodimo vzorec prilagodimo za rešitev problema in ga vstavimo v celotno rešitev. V večini primerov je potrebno vstaviti več različnih vzorcev, da dobimo neko novo rešitev.

6.1.3 Preverjanje rešitve

Ko imamo pripravljeno rešitev moramo preveriti ali je ta pravilna. Pogled na preverjanje pravilnosti rešitve je lahko teoretične in praktične narave. Razhroščevanje (*ang. debugging*) spada me vrsto aktivnosti, ki nam pomaga pri ugotavljanju pravilnosti rešitve. Splošno velja da proces razhroščevanja, z programom, ki nam pomaga razhroščevati (ang. debugger) ali brez njega, poglablja razumevanje računalniške znanosti. Z tem ko učenci razmišljajo, kako bodo preverjali ali njihov program deluje pravilno, hkrati v njih poteka miselni proces refleksije o tem kako so implementirali določen program in kako ga bojo morebiti morali spremeniti.

Na nivoju do srednje šole uporabljamo praktične metode ugotavljanja pravilnosti programa, kot je razgroščevanje. Ko želimo znanje pravilnosti delovanja poglobiti se lahko lotimo tudi teoretične amalize.

6.1.4 Refleksija

Refleksija je mentalni proces ali obnašanje, ki nam omogoča da neko delovanje analiziramo in o njem tudi premislimo. Refleksija je pomembno orodje v splošnem učnem procesu, prav tako spadam med kognitivne procese višjega reda. Z refleksijo učenec dobi priložnost, da stopi korak nižje in premisli o svojem razmišljanju in tako izboljša veščino reševanja problemov. Refleksivno razmišljanje je proces, ki zahteva veliko časa in vaje. Med procesom reševanja problemov, lahko refleksijo uporabimo na različnih stopnjah.

- *Pred* reševanjem problemom. Ko problem preberemo, in že načrtujemo rešitev, se splača

uporabiti refleksijo in razmisliti o tem ali smo morda že reševali podoben problem in temu primeren vzorec algoritma.

- *Med* reševanjem problema. Ko rešujemo problem refleksija služi, kot pregled, kontrola in nadzor. Na primer, ko nastopijo težave pri načrtovanju rešitve ali morda zaznamo težavo ali napako. Temo procesu lahko pravimo **refleksija v akciji**.
- *Po* reševanju problema. Ko že najdemo rešitev, ki deluje, nam refleksija služi kot orodje z katerim pregledamo učinkovitost delovanja. Pregledamo strateške odločitve, ki so bile sprejete med samim načrtovanjem rešitve.

Refleksija je kreativni proces in je pomemben za učenca tako kot za učitelja.

7 METODE IN STRATEGIJE PRI UPORABI SPLETNIH PORTALOV

Primer strategij in metod spletnega portala za učenje **Jave**. [8]:

- Scaffolding -> Gradnja študentovega znanja pri katerem pomaga mentorja, z svojim znanjem in izkušnjami.
- Bloomova taskonomija. Zakaj je pomembno vključevanje Bloomove taksonomije in kako jo vključujemo.
- Konstruktivizem: Aktivnost študentov pri gradnji znanja. Učenje z eksperimentiranjem. Problemski pristop.

Kaj od katerih metod predstavlja v uporabi spletnega portala . . . :

- Spletni portal -> Scaffolding + Bloom
- Naloge narejene tako, da podpirajo konstrutivno metoIn tudi nekatere slabosti, če hih najdem v literaturi -> problemski pristopom

V naslednjem poglavju sledi pregled tehnik aktivnih metod poučevanja. V poglavju sledi obravnava didaktičnih pripomočkov, oblik pouka, in projektno delo [7].

7.1 DIDAKTIČNI PRIPOMOČKI

Med didaktičnimi pripomočki najdemo številna orodja:

- **pedagoške igre**,
- računalništvo brez računalništva,
- **bogate naloge**,
- miselni vzorci,
- klasifikacija,
- metafore.

V povezavi z spletnimi portali za učenje programiranja nas bodo zanimale le nekatera.

7.1.1 PEDAGOŠKE IGRE

7.1.2 BOGATE NALOGE

7.2 RAZLIČNE ORGANIZACIJSKE OBLIKE POUKA

Računalniško znanost lahko poučujemo tako, da jo predavamo, vendar to ni v skladu z naprednimi nazori poučevanja aktivnega učenja, ki smo ga do sedaj spoznali. Za uspešno in koristno učenje se moramo temu pristopu čim bolj izogniti. To velja predvsem za izobraževanje na nivoju **OŠ** in seveda tudi **SŠ**. Kot smo že poudarili v poglavju? je pomemben aktivni pristop v učnem procesu.

Pomembno je tudi v kakšni obliki dela poteka pouk. V nasprotju z frontalnim delom, lahko pouk organiziramo na naslednje načine.

Samostojno delo: Prvi način je morda najenostavnejši za organizacijo dela v učilnici in omogoča aktivno učenje za vse učence. Taka oblika organizacije je primerna predvsem, ko učitelj želi preveriti ali vsi učenci sledijo in znajo uporabljati določeno znanje in veščine, kot je na primer uporaba integriranega razvojnega okolja (*ang. Integrated Development Environment (**IDE**)*) ali sledenje določenemu algoritmu.

Delo v parih: Razred razdelimo v pare, ti rešujejo programerske ali ne programerske naloge. V primeru programerskih nalog, učenca, ki sta v paru rešujeta programersko nalogo

tako da je eden v vlogi **voznika** in drugi v vlogi **navigatorja**. Prvi, voznik ima v nadzoru tipkovnico in miško. Drugi sledi razvojnemu procesu in analizira napredek skupaj z voznikom. Oba seveda zamenjujeta vloge. Programiranje v parih vodi v proces reševanja problemov na dveh nivojih, en nivo predstavlja nalogo kodiranja, drugi predstavlja uporabo strategij pri reševanju problema. Ko so naloge niso programerske in jih ne izvajamo na računalniku, zgubimo nalogo voznika. Kljub temu lahko izvajamo tako obliko pouka, saj lahko sklepamo, da je delo v parih, pri reševanju problemov, primernejše kot v večjih skupinah, kjer obstaja večja možnost, da nekateri učenci dominirajo v skupini in teko druge učence prikrajšajo za sodelovanje.

Skupinsko delo Druga oblika organizacije je delo v skupi ali timu in je primerna v naslednjih primerih:

a. ko sta potrebna več kot dva učenca za opravilo neke naloge,
b. Ko učitelj želi izkoristiti raznolikost v skupini,
c. ko je razred razmeroma velik in si učitelj želi olajšati delo tako da učence razdeli v manjše skupine,
d. ko želi da so vključeni vsi učenci, a le eden iz posamezne skupine naj bi predstavljal narejeno delo.

Skupinsko delo - sestavljanka (*ang. Jigsaw Classroom*) Po navodilih spletne strani razredne sestavljanke (*ang. Jigsaw classroom*) je oblika organizacije na naslednji način.

1. Učence razdelimo na skupine 5 - 6. Vsak od njih ima nalogo, da predela posamezno nalogo, poglavje, ki je razdeljeno na toliko pod poglavij ko je učencev v skupini. Vsak od njih je odgovoren, da se nauči posamezno podpoglavje in to znanje posreduje naprej drugim učencem.
2. Preden učenci preidejo k poročanju posameznega podpoglavja, se sestanejo z učenci drugih skupin, ki imajo isto nalogo oz podpoglavje. S tem zagotovimo večjo točnost naučenega.
3. Učenci se vrnejo nazaj v svoje prvotne heterogene skupine, in poučijo svoje sošolce o tem kaj so se naučili.

Učitelj se odloči kaj po končni izdelek, ali bo to napisano kratko poročilo, ali plakat, ali kak drugi pisni izdelek, delo se lahko zaključi tudi brez končnega izdelka.

Kot je razvidno z organizacije dela **sestavljanke** so prednosti ogromne, tiste ki omogočajo kognitivni razvoj in tiste, ki socialnega. Učenje v tej obliki vzpodbuja učenje, poslušanje, sodelovanje in deljenje znanja.

7.3 Programiranai pouk

7.4 Projektno delo

Preglejmo najprej nekatere lastnosti, ki jih prinaša projektno delo. To lahko poteka tako, da učenci delajo samostojno ali v skupini. Učitelj je tisti, ki vodi proces projektnega dela. Učenec je pri projektnem delu bistveni člen in mu tako omogoča aktivno učenje.

7.5 Učne strategije

7.5.1 Aktivno učenje in model aktivnega učenja

Vsak pouk računalništva mora biti zgrajen kot model in bi moral upoštevati naslednja načela:

- Vzpodbujati mora študente z pozitivno naravnanim poukum in omogočati mora okolje kjer študent najde pomoč.
- Pouk računalništva je grajen na konstruktivnih metodah poučevanja in aktivnem učenju.

Konstruktivizem je kognitivna teorija, ki preučuje naravo procesov učenja. Po tem principu naj bi učenci konstruirali novo znanje na osnovi preurejanja in izpopolnjevanja že obstoječega znanja. Znanje se gradi na obstoječih mentalnih strukturah in na odzivu, ki ga dobi učenec iz učnega okolja. Mentalne strukture so grajene korak za korakom, ena za drugo, seveda s to metodo lahko pride tudi do sestopanja ali slepih koncev. Proces je povezan z Piagetovim mehanizmom asimilacije [7].

Pri **aktivnem učenju** je najpomembnejše to, da učenci z lastno aktivnostjo ugotovijo, sami za sebe kako nekaj deluje. Sami si morajo izmisliti primere, preiskusiti lastne veščine in reševati neloge, ki so jih že ali jih še podo spoznali. Učenje je aktivno usvajanje, je gradnja idej in znanja. Za učenje mora biti posameznik aktivno vključen v gradnjo svojih lastnih mentalnih modelov.

Model aktivnega učenja je sestavljen s štirih korakov [7].

- **Sprožilec** Je je naloga, ki predstavlja iziv za uvod v novo tematiko. //Gerlič -> Motivacija.
- **aktivnost** Študenti izvajajo aktivnost, ki jim je bila predstavljena v sprožilcu. Ta kora je lahko kratek ali lahko zavzame večju del učne ure. To je odviso od vrste sprožilca in izobraževalnih ciljev.

- **diskusija** sledi po koncu aktivnosti, kjer se zbere zeloten razred, neglede na obliko dela. V temo koraku študenti izpopolnijo koncepte in ideje, kod del konstruktivnega učnega procesa.
- **povzetek** je lahko izračen v različnih oblikah, kot so zaogrožene definicije, lahko so miselni vzorci ali povezav med temami, ki so jih obravnavali študenti in med drugimi temami, ki se navezujejo nanje.

Ko se ta model izkaže za primernega, ga lahko uporabimo v številnih učnih urah v različnih variacijah.

7.5.2 UČENJE NA DALJAVO

7.6 TIPI NALOG

7.6.1 ZAPOLNI PRAZNA MESTA

Tip nalog začenniku ponuja ogrodje programa, del programske kode, na katerem dijak usvoji novo znanje in/ali lahko uporablja že pridobljeno znanje.

8 KATEGORIZIRANJE SPLETNIH PORTALOV

8.1 VRSTA VSEBINE

Po hitrem pregledu izbranih spletnih portalov lahko ugotovimo, da je

8.2 Programski jeziki

9 Ovrednotenje izbranih spletnih portalov in njihove posebnosti

9.1 Pogoji za ožji izbor spletnih portalov

9.2 Določitev Kriterijev

Pri vrednotenju spletnih portalov bomo upoštevali naslednje kriterije,

- število programskih jezikov,
- zahtevano predznanje uporabnika,
- interaktivna povratna informacija,
- problemski pristop,
- jezik spletne strani.

10 Možni načini uporabe spletnih portalov pri pouku

LITERATURA IN VIRI

[1] Martina Fefer, *Uporaba informacijske-komunikacijske tehnologije v osnovnih šolah s prilagojenim programom*, Univerza v Mariboru - Fakulteta za naravoslovje in matematiko, Maribor, 1999. Pridobljeno 4.4. 2016, iz `http://student.pfmb.uni-mb.si/~dgunze/diplomske/d2/s6.html`.

[2] Gerlič, Ivan, *Sodobna informacijska tehnologija v izobraževanju*, DZS, Ljubljana, 2000.

[3] Klemenčič M., *Uporaba računalniškega programa "Postani matematični mojster" pri pouku matematike.*, Pedagoška fakulteta, Ljubljana, 2011.

[4] Anthony Robins, Janet Rountree, and Nathan Rountree, "Learning and Teaching Programming: A Review and Discussion" v *Comuper Science education*, vol 13, No. 2, 2003, pp. 137 - 172.

[5] S.C. Ng, S.O Choy, R. Kwan, S.F. Chan, "A Web-Based Environment to Improve Teaching and Learning of Computer Programming in Distance Education", *ICWL'05 Proceedings of the 4th international conference on Advances in Web-Based Learning*, 2005

[6] L. Ma, J. D. Ferguson, M. Roper, I.Ross, M. Wood, "A web-based learning model for improving programming students' mental models", v *Proceedings of the 9th annual conference of the subject centre for information and computer sciences*, HE Academy, 2008 pp. 88-94.

[7] O. Hazzan, T. Lapidot, N. Ragonis, *Guide to Teaching Computer Science*, Springer, 2011.

[8] Nghi Truong, *A web-based programming environment for novice programmers*, Queensland University of Technology, Australia, 2007.