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.

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

Sem prištevamo vse tiste aktivnosti, s katerimi želimo udeležence izobraževanja usposobiti na različnih ravneh tako, da se bodo ti z računalnikom in informacijskimi sistemi ukvarjali na poklicni ravni [2].

Gerlič raziskuje podrobno strokovno področje na katerem se izobražujejo bodoči učitelji računalništva. Lahko povemo, da sem spadajo vse ozko usmerjene računalniške in informacijske smeri srednjih šol, visokih ter univerzitetnih študijev. Zanima nas področje programiranja, kar je predvsem predmet strokovnega izobraževanja. Čeprav se bomo v diplomskem delu posvetili spletnim portalom za učenje programiranja na splošnem nivoju, bomo pozneje vso znanje zlahka prenesli tudi na ožje strokovno izobraževanj, saj je tu uvod, začetkov programiranja, podoben na vseh težavnih stopnjah.

2.3 Programiranje v OŠ

Pouk računalništva v OŠ poteka kot izbirni predmet ali kot neobvezni izbirni predmet. Za začetek bomo navedli kako je definirano Računalništvo v učnem načrtu za izbirne predmete v osnovni šoli [9]: *"Računalništvo je naravoslovno-tehnični izbirni predmet, pri katerem se spoznavanje in razumevanje osnovnih zakonitosti računalništva prepleta z metodami neposrednega dela z računalniki, kar odpira učencem in učenkam možnost, da pridobijo tista temeljna znanja računalniške pismenosti, ki so potrebna pri nadaljnjem izobraževanju in vsakdanjem življenju. ".*

2.3.1 Izbirni predmet računalništva

Izbirni predmeti računalništva so sestavljenih z treh predmetov in jih učenke in učenci lahko izberejo v tretjem trilerju, v 7., 8. in/ali 9. razredu. Prvi od teh predmetov je **računalništvo - urejanje besedil**, kjer si učenci pridobijo osnovna znanja, ki so potrebna za razumevanje in temeljno uporabo računalnika. Naslednja dva predmeta sta **računalniška omrežja** in **multimedija**, kjer se ta znanja spiralno nadgradijo.

Zanima nas kje se pri teh predmetih računalništva pojavlja programiranje. Vsak izmed teh predmetov ima operativne učne cilje tako razdeljeno, da programiranje najdemo v tretji enoti, ki spada med dodatne vsebine. Posamezne operativne cilje, dejavnosti in vsebino prikazuje tabela 1.

Tabela 1: Operativni cilji, dejavnosti in vsebine izbirnega predmeta računalništvo za III. dodatno enoto. [9]

OPERATIVNI CILJI	DEJAVNOSTI	VSEBINE
• napisati algoritem z odločitvijo, ki reši preprost vsakdanji problem; • izdelati in spremeniti računalniški program z odločitvijo.	• analizirati preprost problem; • uporabljati osnovne korake programiranja.	• risanje diagrama poteka za problem z odločitvijo; • izdelava računalniškega programa.

.

Programiranje se pojavlja le kot dodatna vsebine, kar se kaže v uresničitvi smeri računalništva, ki zagovarja **splošno usposobljenost**, kot smo jo opredelili v poglavju 2.1. Vendar se tu učiteljem računalništva ponuja izjemna priložnost, da z učenci in učenkami naredi korak v osnove programiranja.

2.3.2 Neobvezno izbirni predmet računalništva

Opredelitev predmeta v učnem načrtu [10], pravi, da neobvezni izbirni predmet računalništva učence seznanja z različnimi področji računalništva, računalniškimi koncepti ter procesi in jih ne učijo dela z posameznimi programi. Učenci se seznanjajo tehniko in metodami reševanja problemov, razvijajo algoritmičen način razmišljanja. Opredelitev zadostuje prvi smeri računalniške pismenosti, ki zagovarja **sposobnost računalniškega programiranja.**. Ker se v tej diplomski nalogi še posebej posvečamo programiranju nam učni načrt tega predmeta dosti bolj ustreza in si ga bomo podrobneje pogledali, saj nam bo pregled operativnih ciljev služil kot vodilo pri nadaljnjem delu. Najprej preglejmo splošne cilje, katere povzemamo po učenem načrtu [10] in jih morajo doseči učenci:

- spoznavajo temeljne koncepte računalništva,
- razvijajo algoritmični način razmišljanja in spoznavajo strategije reševanja problemov,
- razvijajo sposobnost in odgovornost za sodelovanje v skupini ter si krepijo pozitivno samopodobo,
- pridobivajo sposobnost izbiranja najustreznejše poti za rešitev problema,
- spoznavajo omejitve človeških sposobnosti in umetne inteligence,
- se zavedajo omejitev računalniških tehnologij,
- pridobivajo zmožnost razdelitve problema na manjše probleme,
- se seznanjajo z abstrakcijo oz. poenostavljanjem,
- spoznavajo in razvijajo zmožnost modeliranja, strokovno terminologijo.
- razvijajo ustvarjalnost, natančnost in logično razmišljanje,
- razvijajo in bogatijo svoj jezikovni zaklad ter skrbijo za pravilno slovensko izražanje in strokovno terminologijo.

Neobvezni izbirni predmet je namenjen učencem 4., 5. in 6. razreda. V učnem načrtu so operativni cilji predstavljeni tako, da so temeljni označeni **krepko** in izbirni *poševno*. Navedli bomo tiste, ki so navezujejo na programiranje in strategije reševanja problemov. Operativni cilji so razdeljeni na vsebinske sklope.

Vsebina/sklop: Algoritmi

- **razumejo pojem algoritem**,
- **znajo vsakdanji problem opisati kot zaporedje korakov**,
- **znajo z algoritmom predstaviti preprosto opravilo**,
- **algoritem predstavijo simbolno (z diagramom poteka) ali s pomočjo navodil v preprostem jeziku**,
- **sledijo algoritmu, ki ga pripravi nekdo drug**,
- **znajo v algoritem vključiti vejitev (če) in ponavljanje (zanke)**,
- *znajo algoritem razgraditi na gradnike (podprograme)*,

- **znajo povezati več algoritmov v celoto, ki reši neki problem**,
- *razumejo vlogo testiranja algoritma in vedo, da je testiranje orodje za iskanje napak in ne za potrjevanje pravilnosti,*
- *primerjajo več algoritmov za rešitev problema in znajo poiskati najustreznejšega glede na dana merila,*
- *znajo uporabiti nekatere ključne algoritme za sortiranje in iskanje,*
- *poznajo osnovne algoritme za iskanje podatkov.*

Vsebina/sklop: Programi

- **znajo slediti izvajanju tujega programa**,
- **znajo algoritem zapisati s programom**,
- **znajo v program vključiti konstante in spremenljivke**,
- **razumejo različne podatkovne tipe in jih znajo uporabiti v programu**,
- **znajo spremenljivkam spremeniti vrednost s prireditvenim stavkom**,
- **znajo v programu prebrati vhodne podatke in jih vključiti v program**,
- **znajo izpisovati vrednosti spremenljivk med izvajanjem programa in izpisati končni rezultat**,
- **v program vključijo logične operatorje**,
- **znajo uporabiti pogojni stavek in izvesti vejitev**,
- **razumejo pojem zanke in ga znajo uporabiti za rešitev problema**,
- *razumejo kompleksnejše tipe podatkov (nizi, seznami/tabele) in jih znajo uporabiti v programu,*
- **prepoznajo in znajo odpraviti napake v svojem programu**,
- *znajo popraviti napako v tujem programu,*
- *znajo spremeniti program, da dosežejo nov način delovanja programa,*
- *znajo rezultate naloge zapisati v datoteko,*
- **se seznanijo z dogodkovnim programiranjem**,
- **so zmožni grafične predstavitve scene (velikost objektov, ozadje, pozicioniranje)**,
- **so zmožni sinhronizacije dialogov/zvokov**,
- **so zmožni razumeti in realizirati interakcije med liki in objekt**i,
- **so zmožni ustvarjanja animacij**.

Vsebina/sklop: Podatki

- **razlikujejo podatek in informacijo**,
- **razumejo dvojiški sistem zapisovanja različnih podatkov**,
- **razumejo kodiranje podatkov**,
- **razumejo, da obstajajo podatki v različnih pojavnih oblikah (besedilo, zvok, slike, video)**,
- *poznajo načine predstavitev določenih podatkov in odnose med njimi (dvojiška drevesa*

in grafi),
- *vedo za stiskanje podatkov in vedo, da je stiskanje lahko brez izgub ali z izgubami*,
- **pojasnijo razliko med konstantami in spremenljivkami v programu**,
- *poznajo osnovne algoritme za iskanje podatkov*,

Vsebina/sklop: Reševanje problemov

- *znajo uporabiti različne strategije za reševanje problema*,
- **znajo našteti faze procesa reševanja problema**,
- *znajo postavljati vprašanja in ugotoviti, kateri podatki so znani*,
- *znajo za podano nalogo izluščiti bistvo problema*,
- **znajo najti ustrezno orodje, s katerim rešijo problem**,
- **znajo problem razdeliti na več manjših problemov**,
- **znajo načrtovati in realizirati rešitev**,
- *za podano rešitev znajo oceniti posledice in vpliv na "okolje"*,
- *znajo uporabiti znano strategijo v novih okoliščinah*,
- *znajo ustvariti nov algoritem za bolj kompleksne probleme*,
- **znajo učinkovito sodelovati v skupini in rešiti problem z uporabo informacijsko-komunikacijske tehnologije znajo ceniti neuspešne poskuse reševanja problema kot del poti do rešitve**,
- *znajo kritično ovrednotiti rešitev in ugotoviti ali rešitev uspešno reši dani problem*,
- *znajo kritično ovrednotiti strategijo reševanja problema*,
- *zavedajo se omejitev informacijsko-komunikacijske tehnologije pri reševanju problemov*.

Vsebina/sklop: Komunikacija in storitve

- **znajo uporabiti ustrezna orodja in metode za iskanje po spletu**,
- **znajo uporabiti različne iskalne strategije v iskalnikih**,
- **poznajo omejitve pri rabi na spletu najdenih informacij (zavedajo se pojma intelektualna lastnina)**,

Kot smo že povedali smo nekatere cilje izvzeli, čeprav je znanje računalniških omrežij pomembno, ga z vidika programiranja lahko izpustimo. Lahko povemo, da nam je večina operativnih ciljev ostala, saj smo jih lahko izključili le malo. Pridemo do spoznanja, da se v računalniški znanosti pretežen del znanja vrti okoli programiranja in reševanja problemov. Z samih ciljev lahko spoznamo tudi samo zahtevnost snovi. Lahko si upamo napovedati, da jih bomo s pravimi orodji kot so spletni portali za učenje programiranj tudi lažje dosegli.

2.4 Programiranje v SŠ

Pregled: informatika - Gimnazija Pregled: računalništvo - Tehnična gimnazija.

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 Računalniška znanost in programiranje

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

4.1 Osnovni pojmi

Preden nadaljujemo moramo razjasniti nekaj osnovnih pojmov, ki se pojavljajo računalniški znanosti.

4.1.1 Program

Računalniški program je zbirka navodil, ki opravlja točno določeno nalogo in jo izvajamo na računalniku. **Centralno procesna enot**a je tista, ki izvajanje programa omogoča. Računalniški program navadno napiše **programer** v nekem **programskem jeziku**, postopku pisanja programa pravimo **programiranje**. Programski jezik omogoča, da je program zapisan v takšni obliki, da je berljiv za ljudi in je zapisan v **izvorni kodi**. Da računalnik razume napisan program, ga prevede **prevajalnik (*ang. compiler*)** v **strojno kodo.** [11].

4.1.2 Algoritem

Z besedo algoritem ponazarjamo postopek, ki je zgrajen z posameznih operacije in se izvajajo po posameznh korakih. Algoritem daje rešitev za izračune, procesiranje podatkov, avtomatizacijo postopkov. Ime besed *"algoritem"* prihaja z imena **Al-Khwārizmī**, Perzijskega matematika, astronoma, geografa in učenjaka [12].

Pri pregedu učnih načrtov v poglavju 2.3.2 smo lahko zasledili cilje kot so:

- **znajo vsakdanji problem opisati kot zaporedje korakov**,
- **znajo z algoritmom predstaviti preprosto opravilo**,
- **algoritem predstavijo simbolno (z diagramom poteka) ali s pomočjo navodil v preprostem jeziku**.

Prikazali bomo več različnih predstavitev algoritma, najprej bomo rešitev zapisali v pesedilni obliki, zatem bomo rešitev podali z diagramom poteka, na koncu bomo rešitev podali še z

programskim jezikom Scratch in Python. Za primer bomo prikazali primer algoritma, ki reši naslednjo nalogo.

Primer 1: Algoritem - Najdi največje število

Algoritem med podanimi števili poišče največje število.

Postopek zapisan v **besedilu**:

1. Zapišemo si začetno število, naj bo 0.
2. Števila v seznamu preglejujemo po vrsti.
3. Vsakič, ko najdemo večje število, ko je naše začasno število,
4. To začasno število prečrtamo in napišemo tisto z seznamoa, ki je večje.
5. Postopek ponavljamo, dokler ne pridemo na konec seznama.

Podprogram v **Scratchu**:

Podprogram v **Pythonu**:

```
1 def najdiNajvecjeStevilo(seznam):
2     """Funkcija poisce najvecje stevilo v seznamu."""
3     i = 0
4     najvecje_stevilo = 0
5     for i in range(len(seznam)):
6         if seznam[i] > najvecje_stevilo:
7             najvecje_stevilo = seznam[i]
8     return najvecje_stevilo
```

4.1.3 Programiranje in kodiranje

Izraz programiranja smo že spoznali. Veliko krat slišimo tudi izraz, **"kodiranje" (*ang. coding*)**. Povedali bi lahko da izraz pomeni, pisanje programske kode, konček izdelek, tisto kar na koncu damo **prevajalniku**, kar je **izvorna koda**, da prevede v **strojno kod**o, katero lahko potem zaganjamo.

Za vsakim programiranjem stoji seveda nek programer, lahko bi rekli, da je za vsakim kodiranjem nekdo, ki mu pravimo **koder**. Programer in koder stav veliko krat, dana v isti koš, vendar si to čisto ne zaslužita, saj je programer, nekdo ki načrtuje rešitve, na različne načine in z razčličnimi orodji preden sploh zapiše kaj programske kode. Koder po drugi strani je tista oseba, ki se dobro spozna na programske jezike vendar, dela veliko krat po načrtu programera, je tisti ki na kocu zapiše rešitve in je pri tem zelo unčikovita. Čeprav se ta dva poklica zelo povezana in so meje med njima tudi veliko krat zabrisane sploh, če je programer in koder ena in ista oseba [13].

Pri samem poučevanju računalništva, lahko menimo, da je v prvi vrsti pomembno to, da se učimo strategije reševanja problemov, torej kako poučevati, da bo čim več učencev postalo dobrih programerjev.

4.2 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].

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 programske paradigme objektnega programiranja, saj je ta v zadnjih letih naj bolj popularna.

4.2.1 Objektno orientirano programiranje

V objetno orientirano **OO** programiranje je način kako programerji razmišljajo o svojem delu. Princip OO model realnosti sveta predstavlja v **razredih *ang. class***. Z takim načino zapisa programske kode je ramišljanje o programu dosti bolj naravno [14].

Objekt je predstavnik različnih stvari, ki jih želimo predstavljati v programskem razredu. Te stvari so lahko kar koli, od realnih objektov in vse do konceptov. Podajmo primer objekta mačke. Mačka ima številne karakteristike, kot so barva, ime, teža . . . , tem lasnostim pravimo

da so lasnosti objekta. Mačka je živo bitje, zato njeno početnje, kot je mjavkanje, spanje, igranje ..., dejanjem mače pravimo, da so metode objekta. Pri objetih lahko uporabimo analogijo in objekte poimenujemo z samostalniki, metode so glagorli in vrednosti lasnosti objekta so pridevniki. V nadaljevanju si bomo ogledali nekatere značilnosti, ki definirajo ne programski jezi kot OO [15].

Razred (*ang. Class*): V resničnem življenju lahko objekte združujemo po nekih določenih kriterijih. Orel in sinička sta oba ptiča, zato jih lahko damo skupaj v razred katerega poimenujemo Ptiči. Razredi so načrti ali recepti za objekte, tako lahko ustvarimo več objektov iz istega razreda, saj je razred le shma.

Enkapsulacij (*ang. Encapsulation)*: je koncept ,ki predstavlja, da so podatki, torej *lasnosti* objektov in opravila, ki jih lahko opravljajo ali *metode* objektov, združeni.

Združevanje (*ang. Aggregation*): pomeni, da lahko več objektov združimo v en objet. To predstavlja močno orodje pri razčljenjevanju problemov na manjše pod probleme.

Dedovanje (*ang. Inheritance*): je eleganten način kako porabimo eno kodo več krat. Podajmo primer, imamo splošen razred Oseba, ta ima lasnosti kot je ime, datum rojstva in ima napisane metode, ki prestavljajo funkcionalnost kot je, da Oseba lahko govori, hodi, je, spi. Zatem bi želeli bolj specifičen razred ko je Programer. Lahko bi vso kodo ponovno napisali in ji dodali specifično za programerja. Dedovanje omogoča, da povemo da Programer deduje od razreda Osebe in si tako prihranimo velik del dela.

Polimorfizem (*ang. polymorphisem*): je način kako lahko isto ime metode uporablja več različnih razredov in posledično objektov neglede nato, da je najverjetneje koda v njem različna.

Programsko paradigmo OO programiranja smo povzeli na kratko, da bi lažje razumeli zakaj je tako popularna. V nadaljevanju bomo spoznali, da je večina programskih jezikov, ki se uporabljajo dan danes OO ali vsaj vsebijejo nekaj lasnosti OO programskih jezikov.

4.3 Programski jeziki

V tem poglavju bomo povzeli osnovne značilnosti posameznij programskih jezikov. Če na spletu v spletnem iskalniku podamo zahtevo po najpopularnejšuh programskih jezik, dobimo podobne rezultate večih spletnih strani[1] [2][3] in sicer: **JAVA, C, C++, Python, C#** v top 10 za nas pomembne majdemo še **Java Script**.

Programski jeziki v izobraževanju so se skozi zgodovino menjavali, tako kot se je rezvijala računalniška znanost, kar smo že povzeli v poglavju 3. Zanima nas, kateri so prograsmki jeziki,

[1] Pridobljeno 27.04.2016 iz, `http://www.tiobe.com/tiobe_index`.
[2] Pridobljeno 27.04.2016 iz, `http://githut.info/`.
[3] Pridobljeno 27.04.2016 iz, `http://pypl.github.io/PYPL.html`.

ki so najbolj primerni za uporabo učenja programiranja in se uporabljajo danes.

Vsak programski jezik bomo z kratim primerom tudi predstavili z primerom programske, kode tako bomo dobili lažjo predstavo kakšna je razlika v sintaksi.

Večina od zgoraj naštetih programskih jezikov je **OO** razen izjeme **C**-ja, ki je predhodnik **C++**.

Za nas bodo z izobraževalnega vidika zanimivi predvsme tisti, ki se uporabljajo pri nas.

4.3.1 Java

Java je več namenski programski jezik, njegova osnov so *razredi* in je OO. Njegova glavna prednost je, da napisano kodo lahko zaganjamo ne glede na platformo na kateri teče, torej so napisani programi neodvisni od operacijskega sistema, ki ga poganja računalnik. Zato je za vsako platformo prilagojen **virtualni stroj za Javo** ***(ang Java Virtual Machin (JVM)***, ki prevedeno kodo poganja. Sintaksa programskega jezika je zelo podobna **C++**. Popularnost programskega jezika je še izboljšal zaradi operacijskega sistem za tablice in telefone **Android**, ki prav tako teče na različici (JVM) oz so programi napisani v Javi [17].

V izobraževanju je Java postala zelo priljubljena, prav zaradi zmožnosti, poganjanja programov na različnih platformah. Uporablja se na primarnem področju izobraževanja, predvsem na srednjem in visokem šolskem področju. V sekundarnem področju izobraževanj se je uporabila predvsem za pisanje programske opreme, ki dopolnjuje izobraževanje, kot so fizikalne simulacije (*fizleti*) in podobno. Eden od razlogov, da se je Java na tem področju dobro uveljavila je tudi ta, da omogoča zagon aplikacije s spletnega brskalnika, vendar je za to potrebna instalacija posebnega vtičnika, ki to omogoča. Primer 2 prikazuje sintakso *"Dobrodošel svet"* napisanega v Javi.

Primer 2: Program napisan v javi

```
1 class First {
2    public static void main(String[] arguments) {
3      System.out.println("Dobrodosel svet!");
4   }
5 }
```

4.3.2 C++

C++ je vse namenski programski jezik, ki je OO in je bil zasnovan kot sistemski programski jezik. Večina operacijskih sistemov je danes napisana v kodi C++ in predhodnika C. Kodo

programskega mora prevesti prevajalnik preden jo lahko zaganjamo, za vsako platformo moramo prevajati posebej. C++ velja za najhitrejši programski jezik, z njim lahko opravljamo tako naloge, kot je neposreden nadzor nad polnilnikom, kot tudi vse višje funkcije, ki jih omogoča. Zato velja za enega težje učljivih programskih jezikov. Programiranje v C++ se uči predvsem na višjem in univerzitetnem izobraževalnem nivoju [18].

Primer 3: Program napisan v C++

```
1 // 'Hello World!' program
2 #include <iostream>
3
4 int main()
5 {
6   std::cout << "Hello World!" << std::endl;
7   return 0;
8 }
```

4.3.3 Java Script

Java Script (JS) se je razvil z potrebe po bogatejših in dinamičnih spletnih straneh. Začetek spleta so predstavljali statični dokumenti, ki so bile povezane z hiper povezavami. Skirptni jezik z imenom Java script se je prvič pojavil z spletnim brskalnikom *Netscape 2.0*. Takrat je bilo možno vstavljanje kratkih odsekov kode, ki so spletne strani naredile dinamične. Težnja po standardizaciji skriptnega jezika se je pojavila ko se je na trgu pojavil *Internet explorer 3.0*, saj je ta imel svojo različico sriptnega jezika *JScript*. Sedaj se standardni jezik imenuje **ECMA Script** oz. točneje **ECMA-262**, ki opisuje glavne dele programskega jezika JavaScript brez specifikacij, spletnega brskalnika [15].

Če smo v prejšnjih poglavjih govorili, da sta bila **Java** in **C++** večnamenska jezika, je JS bil eden tistih, ki so tekli znotraj vgrajenega gostiteljskega okolja, kot je spletni brskalnik. Danes imamo tudi okolja, ki omogočajo, da JS teče na strežnih, na namizju in mobilnih napravah. Torej, kljub zgoraj omenjeni omejitvi, postaja prav tako večnamenski skriptni programski jezik. V spodnjem primeru 4 imamo primer programa *"Dobrodošel svet!"*, z **HTML** ogrodjem. Tako programska kodo odpremo v spletnem brskalniku. Del skriptnega jezika se začne z značko `<script type="text/javascript»` `JS koda` `</script>`. Povejmo še to, da se koda programskega jezika ne prevaja, temveč jo poganja **tolmač**.

Primer 4: Program napisan v JavaScriptu + HTML ogrodje

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Some Page</title>
    <script type="text/javascript">
      alert("Hello World!");
    </script>
  </head>
  <body>
    <p>The content of the web page.</p>
  </body>
</html>
```

4.3.4 Python

Python je zelo pogosto uporabljen večnamenski programski jezik. Njegovo kodo podobno kot JS poganja tolmač. Zasnovan je tako, da je koda čim bolj berljiva in njegova sintaksa omogoča, da programske koncepte zapišemo v čim manj vrsticah, kakor bi jih lahko v Javi ali C++. Če so posamezni odseki ali bloki programske kode pri Javi in C++ označeni z zavitimi oklepaji (“”), jih v Pythonu označimo z tabulatorskim zamikom. V Pythonu lahko uresničimo več programskih paradigem, kot je OO ali proceduralno programiranje. Omogočen je dinamičen tip spremenljivk, ima urejeno avtomatsko upravljanje z pomnilnikom in ima veliko standardno knjižnico [19].

Pyon se veliko uporablja tudi v izobraževalne namen. Pri nas se priporoča za začetke učenja programskega jezika na srednjem šolskem izobraževalnem nivoju. Zaredi tega, ga bomo v diplomskem delu uporabljali, ko glavni demonstracijski programski jezik.

Primer 5: Program napisan v Pythonu HTML ogrodje

```
print (‘‘‘Hello world!’’’)
```

4.4 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 2 prikazuje primer spremenljivke.

Tabela 2: 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

V nadaljevanju bomo pregledali in skušali razložiti osnovne koncepte pri programiranju. Z primerom bomo pokazali enega izmed načinov, kako jih prestavimo. Za vodilo bomo uporabili učni načrt za OŠ, ki smo ga pregledali v poglavju 2.3.2 in SŠ, ki je v poglavju 2.4. Primeri programov, ki jih bomo uporabljali in prilagodilo so povzeti s knjige in spletne strani *"Learning python the hard way"* [20].

4.4.1 Spremenljive

V tem poglavju bomo uresničili naslednje cilje vendar smo jim nekoliko spremenili vrstni red:

- **znajo izpisovati vrednosti spremenljivk med izvajanjem programa in izpisati končni rezultat**,
- **znajo spremenljivkam spremeniti vrednost s prireditvenim stavkom**,
- **znajo v program vključiti konstante in spremenljivke**,
- **razumejo različne podatkovne tipe in jih znajo uporabiti v programu**,
- **znajo v programu prebrati vhodne podatke in jih vključiti v program**,

Osnovno interakcijo z računalnikom lahko opišemo na naslednji način. Računalniku damo neke vhodne podatke, ta podatke po navodilu programa obdela in nam poda rezultate na neko izhodno naprava. Ta izhodna naprava je na primer zaslon in na njem se izpisujejo obdelani podatki. Izpišimo nekaj stavkov, pri tem uporabljamo ukaz `print`.

Primer 6: Izpis besedila na zaslon | `01_izpis_na_zaslon.py` **[20]**

Navodilo naloge: Sledi navodilu, ki je zapisano v programski kodi.

```
1 #Del programa, ki nocemo, da ga uposteva tolmac,
2 #oznacimo z # in ga imenujemo komentar.
3 print "Pozdravljeni, to je nas prvi izpis na zaslonu."
4 print "Izpis druge vrstice na zaslonu."
5 print "Izpisovanje na zaslon je zabavno!"
6 print "Izpisemo lahko tudi pravzno vrstico. \n"
7 #Za izpis prazne vrstive uporabimo "\n"
8 print "Pred to vrstico je prazna! in za njo.\n"
```

```
$ python 01_izpis_na_zaslon.py
Pozdravljeni, to je nas prvi izpis na zaslonu.
Izpis druge vrstice na zaslonu.
Izpisovanje na zaslon je zabavno!
Izpisemo lahko tudi pravzno vrstico.

Pred to vrstico je prazna! in za njo.
```

Ena izmed glavnih nalog računalnikov so računske operacije, zato si poglejmo dva primera izračunov v programskem jeziku Python.

Primer 7: Računske operacije | `02_racunske_operacije.py` **[20]**

Navodilo naloge: Sledi navodilu, ki je zapisano v programski kodi.

```
1 #Izračunajmo nasednje izrazein izpišimo njiho rezultat.
2 #Preden poženemo program izračunajmo vrednost sami.
3 print 100 - 5%2 + 3*4 - 22/3
4 print 4+7 > 13
```

```
$ python 01_uporaba_spremenljivk.py
104
False
```

Spremenljivke so način kako shranjujemo podatke v računalniku. Ime spremenljivke ima podobno vlogo kot imena ljudi ali stvari v vsakdanjem življenju. Ljudje in stvari imajo imena zato, da si jih lažje zapomnimo in se z njimi in o njih lažje pogovarjamo. Podobno je to v programiranju, izbrati si moramo dobra imena spremenljivk, saj bomo tako lažje brali napisano kodo. Poglejmo primer 8.

Primer 8: Izpis besedila na zaslon | `03_uporaba_spremenljivk.py` **[20]**

Navodilo naloge:

Na parkirišču je 100 avtomobilov, vsak izmed avtomovilov ima 5 sedežev. Z temi avtomobili želimo pripeljati 90 potnikov od tega jih ima 30 vozniško dovoljenje. Izračunaj in izpiši naslednje podatke.

- Koliko avtomobilov je navoljo. Koliko šoferjev je navoljo?
- Koliko avtomobilov bo ostalo na parkirišču, če bodo vozili vsi šoferji?
- Koliko ljudi lahko prepeljemo z vsemi avtomobili?
- Koliko avtomobilov bomo uporabili, da pripeljemo vse potnike?
- Kakšno je povprečno število potnikov, če vozijo vsi vozniki?

```
1 #1. Določimo spremenljivke:
2 avtomobili = 100
3 prostor_v_vsakem_avto = 5.0
4 potniki = 90
5 soferji = 30
6 #2. Izračunajmo vrednosti in jih shranimo v spremenljivke:
7 avtomobili_ostali_na_parkiriscu = avtomobili - soferji
8 kapaciteta_vsi_avtomobili = avtomobili * prostor_v_vsakem_avto
9 avtomobili_na_st_potnikov = potniki/prostor_v_vsakem_avto
10 povprecno_stevilo_potnikov = potniki/soferji
11 #3. Izpišimo vse zahtevane podatke.
12 print "Na voljo je", avtomobili, "avtomobilov."
13 print "Na voljo je", soferji, "šoferjev."
14 print "Na parkiriscu bo ostalo",avtomobili_ostali_na_parkiriscu, "
     avtomobilov."
15 print "Z vsemi avtomobili lahko prepeljemo",
     kapaciteta_vsi_avtomobili, "potnikov."
16 print "Minimalno stevilo avtomobilov je", avtomobili_na_st_potnikov
     , ",da pripeljemo vse potnike."
17 print "Povprecno stevilo potnikov je",povprecno_stevilo_potnikov, "
     ,ce vozijo vsi soferji."
```

```
$ python 01_uporaba_spremenljivk.py
Na voljo je  100 avtomobilov.
Na voljo je  30 šoferjev.
Na parkiriscu bo ostalo  70 avtomobilov.
Z vsemi avtomobili lahko prepeljemo  500.0 potnikov.
Minimalno stevilo avtomobilov je  18.0, da pripeljemo vse potnike.
Povprecno stevilo potnikov je  3, ce vozijo vsi soferji.
```

4.4.2 Logični operaterji

V tem poglavju bomo uresničili naslednje cilje:

- **v program vključijo logične operatorje**,

4.4.3 Pogojni stavki in vejitve

V tem poglavju bomo uresničili naslednje cilje:

- **znajo uporabiti pogojni stavek in izvesti vejitev**,

4.4.4 Zanke

V tem poglavju bomo uresničili naslednje cilje:

- **razumejo pojem zanke in ga znajo uporabiti za rešitev problema**,

4.4.5 Kompletksi tipi podatkov

V tem poglavju bomo uresničili naslednje cilje:

- *razumejo kompleksnejše tipe podatkov (nizi, seznami/tabele) in jih znajo uporabiti v programu*,

5 Spletni portali za učenje programiranja

Spletne portale za učenje programiranja (**SPUP**) bomo predstavili in spoznali tako, da bomo najprej pregledali kaj so bili glavni razlogi, da se je pojavili razvoj teh na univerzah, ki poučujejo računalniško znanost. Razen tehnoloških zmožnosti IKT za nastanek spletnega portala nas bodo zanimale težave, ki so jih skušali premostiti z SPUP.

Spletni portali za učenje programiranja so nastali na različnih univerzah po svetu. V nadaljevanju bomo pregledali nekaj takšnih primerov. Zanimal nas bo spletni portal, ki je nastal na *odprti univerzi v Hong Kong-u* (**OUHK**), na *Univerze Strathclyde iz Valike britanije* (**US**) in *Queensland University of Technology, Australia* (**QUTA**).

Zanimali nas bodo predmeti, ki veljajo za začetne pri poučevanju računalniške znanosti in programiranja. **Novinci**, kot jih bomo imenovali so študenti, ki se šele začnejo učiti progra-

miranja. V diplomskem delu nas dejanski zanimajo le učenci osnovnih šol in dijaki srednjih, vendar se oni prav tako šele srečujejo s programiranjem, podobno kot študentje in jih bomo zato vse poimenovali **novinci**.

Kot je razvidno z literature bomo lahko sklepali na nekatere skupne značilnosti vseh novincev, ne glede na težavnostno stopnjo na kateri se nahajajo, saj je programiranje veščina, ki ni dana naravno in se je moramo vsak priučiti.

5.1 Razlogi za nastanek spletnih portalov

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, ki ga podrobneje obdelamo v poglavju 7.5.2.

Podobno pravijo na QUTA, saj je pri samem vadenju programiranja 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].

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].

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 (*ang. 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.

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.

Na univerzi QUTA se z začetnimi predmeti programiranja srečujejo s podobnimi težavami, kotna OUHK in US.

1. Inštalacija in nastavitve okolja za programiranje.
2. Uporaba urejevalnika besedil.
3. Razumevanje programskih vprašanj in uporabe sintakse jezika pri pisanju programske kode.
4. Razumevanje napak prevajalnika.
5. Razhroščevanje.

Ugotavljajo, da je pri damih vajah programiranja pomembno, da ob težavah, novinci dobijo čimprajšen odziv mentorja. V velikih razredih se to izkaže za zelo zahtevno. Tudi začetniki, ki uspešno premagajo začetne ovire in se lotijo takojšnjega programiranja, imajo zelo slabo napisano in konstruirano programsko kodo. Pomagati študentom, pistati kvalitetno programsko kodo je prav časovno zelo zahtevno. Težave programiranja se stopnjujejo ko se za učenje progremiranja uporabljajo OO programski jeziki, saj ti zahtevajo visoko stopnjo stopnjo abstraktnega razumevanja programskih konceptov. Za izdelavo spletnega portala za učenje programiranja so na QUTA bili pomembni naslednji cilji [8].

- Omogočiti lažji začetek pri učenju programiranja, z pogostim odzivom mentorjev na težave novincev. S pomočjo ob pravem času spremenimo odnos novincev do programi-

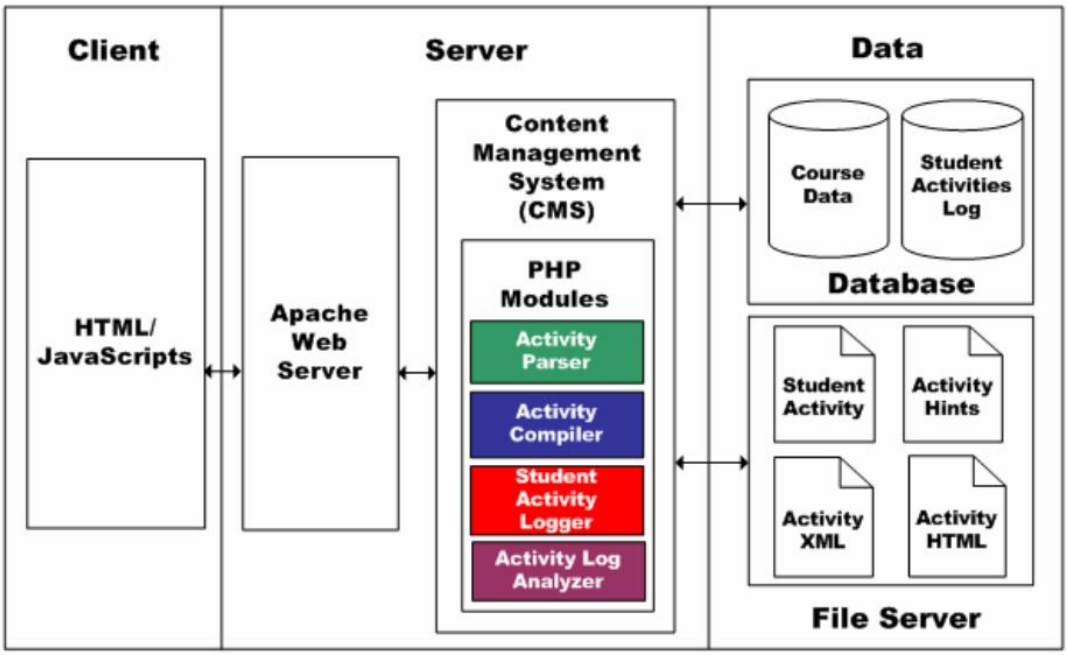
ranja.

- Izboljšati uspeh začetnih predmetov programiranja.
- Da pomagamo mentorjem pri učenju in administraciji predmetov programiranja.

5.2 Primeri implementacije in sistemska arhitektura

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. Študenti na **OUHK** se učijo programiranja v programskem jeziku **JAVA**.

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.



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

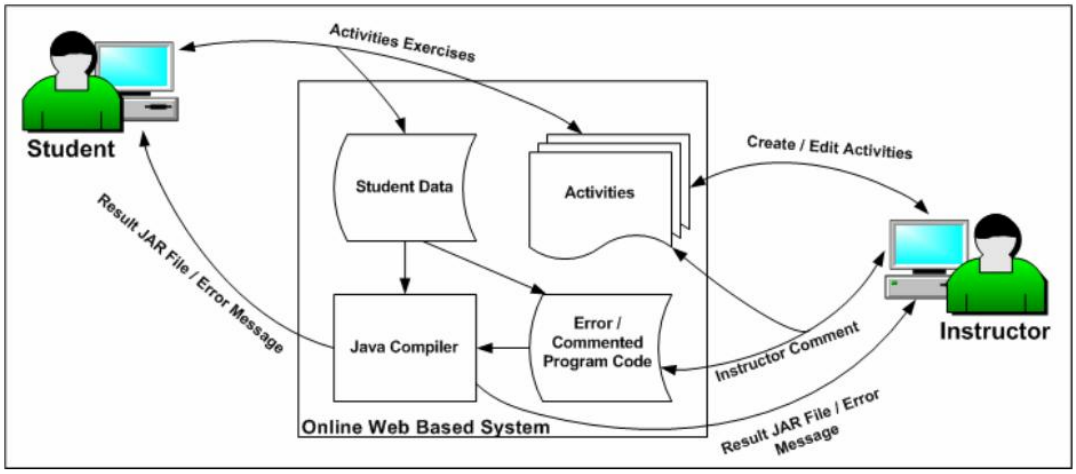
Ti moduli so naslednji, zajem aktivnosti, prevajalnik aktivnosti, dnevnik študentove aktivnosti in analizator dnevnikov aktivnosti. Samo delovanje je naslednje, ko odjemalec pošlje zahtevo za neko aktivnost, se ta naslovi strežniku, ki poišče programsko aktivnost. Z modulom *zajema aktivnosti*, strežnik 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 program-

ske 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.

5.3 Pregled delovanja in interakcija s SPUP

V nadaljevanju opišimo, kako so si zamislili interakcijo med študentom ne OUHK. 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 [5].

Na univerzi v US [6], je okrog strategije kognitivnega konflikta 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.

V nadaljevanju bomo pregledali na kakšen način so si uporabo spletnega okolja zamislili. Spletno učno okolje podpira programski jezik **JAVA**. 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. Torej je vrstni red spoznavanja programskih konceptov 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 [6].

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. Spletni portal v primeru QUTA uči programiranja v programskem jeziku Java in ima naslednjo naslednje zmožnosti [8].

1. Spletni portal za programiranja, ki omogoča naloge tipa “Zapolni prazna mesta”.
2. Ogrodje za analizo, ki preverja kvaliteto in pravilnost, nalog, tipa Žapolni 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 Rezultati izvedenih rešitev SPUP

Večina študentov smeri računalništva na OUHK 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.

Za izboljšanje mentalnih modelov so 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 [6].

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.

Rezultat dela avtorjev spletnega portala QUTA, gre še nekoliko naprej od OUHK in v njihov spletni portal vgradijo, odziv spletnega portala, ki mora o pravilnosti programa in o kvaliteti. Ogrodje *(ang. framework)* za analizo programske kode vsebuje naslednje komponente [8].

- Sintaktično ali semantično opozarjanje na napake ali napake prevajalnika.
- odziv na kvaliteto in pravilnost programske kode.
- Formalni odzin učitelja oz. komunikacija med učiteljem in učencem.

5.5 Kaj so spletni portali z učenje 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].

Strnimo nekatere značilnosti težav novincev.

- Težave pri namestitvi in nastavitvah programske opreme, prevajalnika in razvojnega okolja (OUHK, QUTA).
- Dostop do mentorjev zaradi časovne dostopnosti in Komunikacija v primeru izobraževanja na daljavo (OUHK, QUTA).
- Uporaba urejevalnika besedil (QUTA).
- Razumevanje programskih vprašanj in uprabe sintakse jezika pri pisanj programske kode (QUTA).
- Uporoaba tehnik razhroščevanje (QUTA).
- Razumevanje napak prevajalnika (QUTA).
- Razumevanje osnovnih programskih koceptov, slabo vpliva na reševanje koceptov (US).

TODO: Strnjeno o tem kaj je spletni portal Kaj so spletni portali?

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 analize.

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 Programirana 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 začetniku 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.

[9] Vladimir Batagelj et al., *UČNI načrt, Izbirni predmet: Program osnovnošolskega izobraževanja, Računalništvo*, Ministrstvo za šolstvo, znanost in šport: Zavod RS za šolstvo, Ljubljana, 2002. Pridobljeno 2.4.2016 iz, `http://www.mizs.gov.si/fileadmin/mizs.gov.si/pageuploads/podrocje/os/devetletka/predmeti_izbirni/Racunalnistvo_izbirni.pdf`

[10] Radovan Kranjc et al., *UČNI načrt, Program osnovnošolskega izobraževanja, Računalništvo: neobvezni izbirni predmet*, Ministrstvo za šolstvo, znanost in šport: Zavod RS za šolstvo, Ljubljana, 2002. Pridobljeno 2.4.2016 iz, `http://www.mizs.gov.si/fileadmin/mizs.gov.si/pageuploads/podrocje/os/devetletka/program_razsirjeni/Racunalnistvo_izbirni_neobvezni.pdf`.

[11] Wikipedia contributors, *Computer program*, Wikipedia, The Free Encyclopedia. Pridobljeno 25.4.2016 iz, `https://en.wikipedia.org/wiki/Computer_programming`.

[12] Wikipedia contributors, *Algorithem*, Wikipedia, The Free Encyclopedia. Pridobljeno 25.4.2016 iz, https://en.wikipedia.org/wiki/Algorithm.

[13] Jonah Bitautas, *The Differences Between Programmers and Coders*, Workfunc. Pridobljeno 26.4.2016 iz, http://workfunc.com/differences-between-programmers-and-coders/.

[14] Carl Reynolds, Paul Tymann, *Principles of Computer science*, McGraw-Hill, London, 2008.

[15] Stoyan stefanov, Kumar Chetan Sharman, *Object-Oriented Java Script, Second edition*, Packt Publishing, Ltd, Birmingham, 2013.

[16] Wikipedia contributors, *Object-oriented programming*, Wikipedia, The Free Encyclopedia. Pridobljeno 26.4.2016 iz, https://en.wikipedia.org/wiki/Object-oriented_programming.

[17] Wikipedia contributors, *Java(programming language)*, Wikipedia, The Free Encyclopedia. Pridobljeno 27.4.2016 iz, https://simple.wikipedia.org/wiki/Java_%28programming_language%29.

[18] Wikipedia contributors, *C++*, Wikipedia, The Free Encyclopedia. Pridobljeno 27.4.2016 iz, https://en.wikipedia.org/wiki/C%2B%2B.

[19] Wikipedia contributors, *Python(programming langugage)*, Wikipedia, The Free Encyclopedia. Pridobljeno 30.4.2016 iz, https://en.wikipedia.org/wiki/Python_%28programming_language%29.

[20] Zed A. Shaw, *Learn Python the Hard Way*. Pridobljeno 2.5.2016 iz, http://learnpythonthehardway.org/book/.