INFORMACIJSKA DRUŽBA – IS'2012
12. oktober 2012
VZGOJA IN IZOBRAŽEVANJE V INFORMACIJSKI DRUŽBI

Bolonjska prenova uvodnega programerskega predmeta na Fakulteti za računalništvo in informatiko Univerze v Ljubljani

Bologna-Induced Restructuring of the Introductory Programming Course at University of Ljubljana, Faculty of Computer and Information Science

Luka Fürst
Univerza v Ljubljani, Fakulteta za računalništvo in informatiko
Slovenija
luka.fuerst@fri.uni-lj.si

Viljan Mahnič
Univerza v Ljubljani, Fakulteta za računalništvo in informatiko
Slovenija
viljan.mahnic@fri.uni-lj.si

Povzetek

V pričujočem prispevku predstavljamo nedavno prenovo uvodnega programerskega predmeta na univerzitetnem študiju na Fakulteti za računalništvo in informatiko Univerze v Ljubljani. Prenovo je vzpodbudila bolonjska reforma, ki jo je omenjena fakulteta uvedla v študijskem letu 2009/10. V okviru prenove smo povečali težo sprotnega in samostojnega dela študentov. Za aktivnejšo udeležbo (boljših) študentov pri pedagoškem procesu smo uvedli vsakoletno tekmovanje v programiranju namiznih iger in natečaj za pripravo nalog na temo dedovanja pri objektnem programiranju. Poleg tega smo s pomočjo lastne tehnološke rešitve izboljšali učinkovitost izvajanja vaj. Ocenjujemo, da je prenova predmeta skladna z glavnimi cilji bolonjske reforme.

Ključne besede: univerzitetno poučevanje programiranja, bolonjska reforma, aktivno učenje

Abstract

In this paper, we present our recent restructuring of the introductory programming course at University of Ljubljana, Faculty of Computer and Information Science. The

restructuring was induced by the Bologna reform, which Faculty of Computer and Information Science adopted in the academic year 2009/10. As part of this restructuring, we increased the significance of ongoing and independent student work. To achieve a more active involvement of (better) students in the pedagogic process, we introduced the annual board game programming competition and the annual contest of student problems on the theme of inheritance in object-oriented programming. Using our own technological solution, we improved the way in which laboratory sessions are conducted. We estimate that our restructuring of the introductory programming course conforms to the main goals of the Bologna reform.

Keywords: university-level programming education, Bologna reform active learning

1 Uvod

Uvodni programerski predmet na univerzitetnem študiju marsikje predstavlja precejšnjo oviro za študenta, ki se še ni srečal s programerskimi koncepti in algoritmičnim načinom razmišljanja. Programiranje je težavno tudi z vidika poučevanja (Jenkins, 2002; Milne in Rowe, 2002; Huet *et al*., 2004; Hawi, 2010). Po mnenju Jenkinsa (2002) je programiranje za študente začetnike zahtevno predvsem zato, ker v nasprotju z večino osnovnošolskih in srednješolskih predmetov zahteva povezovanje različnih sposobnosti na različnih ravneh, denimo v okviru hierarhije sintaksa-semantika-struktura-slog ali hierarhije problem-algoritem-program. Na univerzah po svetu se z zahtevnostjo poučevanja programiranja spoprijemajo na različne načine. Čeprav marsikje še vedno uporabljajo izključno tradicionalne pristope, ki poudarjajo osrednjo vlogo učitelja v pedagoškem procesu (angl. teacher-centered learning), se v zadnjem času čedalje bolj uveljavljajo t.i. aktivni pristopi k poučevanju (angl. active learning) (Barak *et al*., 2007; Fernandez *et al*., 2011; Moura in van Hattum-Janssen, 2011), ki poudarjajo učenčevo soodgovornost pri pridobivanju znanja, učitelja pa obravnavajo predvsem kot spodbujevalca (angl. facilitator) učenja. Aktivni pristopi se skladajo s t.i. konstruktivistično teorijo učenja (angl. constructivist learning theory) (von Glasersfeld, 1995). Aktivnejšo vlogo študentov pri pridobivanju znanja zagovarja tudi bolonjska reforma (Wächter, 2004; Fernandez *et al*., 2011), katere cilj je povečati medsebojno primerljivost programov na evropskih univerzah. Bolonjsko reformo smo v študijskem letu 2009/10 uvedli tudi na Fakulteti za računalništvo in informatiko Univerze v Ljubljani. Smernice bolonjske reforme, med katerimi naj omenimo aktivnejšo vlogo študentov v pedagoškem procesu in večji poudarek na sprotnem in domačem delu, smo upoštevali tudi pri poučevanju uvodnega programerskega predmeta na univerzitetnem študijskem programu (ta predmet bomo v nadaljevanju označevali z okrajšavo UPP-FRI). V pričujočem prispevku bomo podrobneje predstavili novosti, ki smo jih kot posledico bolonjske reforme, deloma pa tudi na podlagi lastnih opažanj, uvedli pri predmetu UPP-FRI v študijskih letih od 2008/09 do 2010/11.

V nadaljevanju prispevka bomo najprej (razdelek **Error! Reference source not found.**) podali splošen pregled predmeta UPP-FRI. V razdelku 3 bomo opisali način izvajanja predmeta pred prenovo, v razdelku 4 pa stanje po prenovi. V razdelkih 5, 6 in 7 bomo podrobneje predstavili posamezne elemente bolonjske prenove predmeta UPP-FRI. Z razdelkom 8 bomo prispevek zaključili.

2 Splošen opis predmeta UPP-FRI

Cilj predmeta UPP-FRI (Mahnič in Gams, 2003; Mahnič, 2004) je naučiti študente osnov proceduralnega in objektno usmerjenega programiranja na primeru programskega jezika Java, ki ga poučujejo na mnogih univerzah po svetu (Richards, 2003; de Raadt, Watson in Toleman, 2004; Benander, Benander in Sang, 2004; Anik in Baykoç, 2011). Pri predmetu UPP-FRI se študentje seznanijo z osnovnimi koncepti proceduralnega programiranja (zaporedje stavkov, pogojni stavki, zanke, podprogrami), z osnovami objektnega programiranja, z uporabo enodimenzionalnih in večdimenzionalnih tabel ter tabel objektov, s konceptom dedovanja v objektnem programiranju in z osnovami grafike.Predmet UPP-FRI se izvaja v zimskem semestru prvega letnika v obliki predavanj in vaj. Predavanja potekajo v obliki triurnih[1] tedenskih blokov. V vsakem bloku predavanj predavatelj najprej predstavi nek programerski koncept, nato pa ga ilustrira z množico osnovnih primerov. Predavanja se izvajajo za vse študente prvega letnika univerzitetnega programa hkrati.

Vaje so pred uvedbo bolonjske reforme potekale v blokih po tri ure na teden, nato pa se je njihov obseg zmanjšal na dve uri tedensko. Vaje potekajo v računalniških učilnicah v skupinah po 25–30 študentov pod vodstvom dveh asistentov. Vsak študent ima na voljo svoj računalnik. Obveznosti pri predmetu UPP-FRI so sestavljene iz sprotnega dela in pisnega izpita. Ker pa se je način izvedbe in ocenjevanja obveznosti po bolonjski prenovi precej spremenil, bomo te elemente pedagoškega procesa opisali v naslednjih razdelkih.

3 Predmet UPP-FRI pred bolonjsko prenovo

Pri opisu izvajanja predmeta UPP-FRI pred bolonjsko prenovo se bomo osredotočili na način izvajanja vaj in način ocenjevanja študentov, saj sta to elementa pedagoškega procesa, ki ju je prenova najmočneje spremenila. V letih pred bolonjsko prenovo so vaje potekale v triurnih blokih, od katerih je bil vsak razdeljen na dva dela. V prvem delu, ki je časovno obsegal približno dve tretjini bloka, je eden od asistentov predstavil programersko nalogo in postopoma razvijal njeno rešitev. Študentje so lahko razvoj programa spremljali na projekcijskem platnu in kodo rešitve sproti pretipkovali na svoje računalnike, tako da so jo lahko kasneje sami preizkusili. Drugi del vsakega bloka vaj pa je bil namenjen samostojnemu delu študentov. Pred študijskim letom 2005/06 sta asistenta v tem času pregledovala rešitve naloge, ki so jo študentje reševali doma, v kasnejših letih (do bolonjske prenove) pa so študentje v drugem delu bloka samostojno — z možnostjo manjše pomoči asistentov — reševali nalogo, ki sta jim jo zastavila asistenta. V obeh primerih je študent za opravljeno nalogo prejel točko. Kdor je zbral vsaj 75% možnih točk, je lahko pristopil k pisnemu izpitu v zimskem izpitnem obdobju, preostali študentje pa so morali počakati do poletnega obdobja. Drugih sankcij za neopravljene naloge ni bilo. Pač pa so študentje, ki so želeli boljšo oceno, imeli na voljo tri dodatne (težje) naloge, od katerih je vsaka prinesla določen bonus (do 4 točke) na pisnem izpitu. Pred bolonjsko prenovo je bila končna ocena predmeta določena zgolj na podlagi pisnega izpita, ki je obsegal programerski in teoretični del. Programerski del je bil sestavljen iz petih programerskih nalog v skupni vrednosti 110 točk. Tem točkam so se prištele še točke iz dodatnih domačih nalog (največ 12). Kdor je zbral najmanj 66 točk (60,0%), je bil oproščen teoretičnega dela izpita. Študentje, ki so na programerskem delu zbrali manj kot 50 točk (45,5%), in tisti, ki so prejeli od 50 do 65 točk, a so bili neuspešni na teoretičnem delu, so morali ponovno opravljati celoten izpit. Kot se je izkazalo, opisani način

[1] Povsod v prispevku bomo z besedo “ura” označevali šolske ure, ki trajajo po 45 minut.

poučevanja ni dovolj vzpodbujal sprotnega dela. Samostojnost reševanja sprotnih nalog se ni dosledno preverjala, v precejšnji meri tudi zaradi velikega števila asistentov, ki so imeli pri pregledovanju različne kriterije. Zato neopravljene naloge niso vplivale na oceno predmeta, ampak le na pravico do opravljanja izpita v zimskem izpitnem obdobju. To je imelo za posledico, da se je marsikdo lotil samostojnega programiranja veliko prepozno, da bi si lahko pridobil dovolj znanja za uspeh na izpitu. Te slabosti smo upoštevali pri bolonjski prenovi predmeta, ki jo bomo predstavili v nadaljevanju.

4 Predmet UPP-FRI po bolonjski prenovi

Bolonjska reforma vzpodbuja sprotno delo in sprotno ocenjevanje, zato je bila na fakulteti sprejeta odločitev, da naj bi pri vseh predmetih, kjer je to smiselno, vaje predstavljale pomemben del ocene. Pri predmetu UPP-FRI je ocena po novem sestavljena iz ocene sprotnega dela, s katerim lahko študent zbere največ 50 točk, in ocene izpita, kjer je možno prejeti do 60 točk. Za pozitivno oceno predmeta mora študent zbrati vsaj 50 točk od skupno 110, pri čemer mora vsaj 25 točk prejeti na pisnem izpitu. Pogoje za pristop k izpitu v zimskem izpitnem obdobju smo ukinili, saj je študent, ki slabo opravi vaje, kaznovan že tako, da mora na pisnem izpitu doseči ustrezno večje število točk. Če na vajah ni prejel nobene točke, mora na pisnem izpitu zbrati vsaj 50 točk od 60. Ocena sprotnega dela je sestavljena iz ocen treh programerskih seminarskih nalog (vsaka je vredna po 10 točk) in dveh teoretičnih testov (oba sta vredna po 10 točk). Pisni izpit obsega le še programerski del, ki je po novem sestavljen iz treh nalog, teoretični del učne snovi pa je pokrit že s sprotnima teoretičnima testoma. Seminarske naloge in teoretične teste bomo podrobneje opisali v razdelku 5. Bolonjska reforma zagovarja aktivnejšo udeležbo študentov v pedagoškem procesu. Pri uvodnem programerskem predmetu je študente, ki programiranje že obvladajo, pogosto težko aktivno vključiti v učni proces. Da bi motivirali tudi ta segment študentov, smo uvedli tekmovanja v programiranju namiznih iger in natečaje za pripravo seminarskih nalog na temo dedovanja v objektnem programiranju. Obe novosti bomo predstavili v razdelku 6. Da bi študentje imeli na voljo več časa za domače delo (kar tudi sodi med smernice bolonjske reforme), je fakulteta ob uvedbi reforme zmanjšala obseg vaj s treh na dve uri tedensko. Zato je nastopila potreba po boljšem časovnem izkoristku vaj. Ugotovili smo, da nam veliko časa in energije odvzema pretipkovanje programske kode, ki jo razvija asistent. Študentje so se pri pretipkovanju motili in so zato pogosto potrebovali premor ali pomoč, poleg tega pa zaradi osredotočenosti na pretipkovanje niso mogli dovolj zbrano spremljati razvoja programa. Za rešitev tega problema smo razvili programski sistem *Asistentov asistent*, ki je avtomatiziral prenos programske kode z asistentovega računalnika na računalnike študentov in tako odpravil potrebo po pretipkovanju. Sistem *Asistentov asistent* bomo podrobneje opisali v razdelku 7. Tudi vaje po bolonjski prenovi izvajamo nekoliko drugače. Trije bloki vaj so v celoti namenjeni ocenjevanju seminarskih nalog, preostali bloki pa se izvajajo tako, da asistent predstavi programersko nalogo, nato pa razvije njeno rešitev v sodelovanju s študenti. Praviloma asistent napiše nek del naloge, nato pa študentje razvijejo preostanek. Na nekaterih vajah na kooperativen način rešimo več manjših nalog, na nekaterih pa eno obsežnejšo. Tovrstni način reševanja nalog, ki ga močno olajšuje prej omenjeni sistem *Asistentov asistent*, je skladen z bolonjsko zamislijo o aktivnejši udeležbi študentov pri pedagoškem procesu.

5 Seminarske naloge in teoretični testi

Sprotne seminarske naloge in teoretične teste smo uvedli zato, da bi lahko zadostili bolonjski smernici, ki vzpodbuja večjo utežitev sprotnega dela. Seminarske naloge so obsežnejše programerske domače naloge. Študentje morajo rešiti in zagovoriti tri seminarske naloge, ki so približno enakomerno razporejene po semestru. Prva pokriva osnovne programske konstrukte, druga tabele, tretja pa objektno programiranje s poudarkom na dedovanju. Študentje imajo za realizacijo vsake seminarske naloge po en teden časa. Ker seminarske naloge predstavljajo dobršen del ocene predmeta (30 možnih točk od 110), smo se morali domisliti ustreznega načina njihovega ocenjevanja. Namesto da bi ocenjevali rešitve, ki jih študentje pripravijo doma, ocenjujemo nadgradnje rešitev, ki jih morajo študentje realizirati v omejenem času v okviru zagovorov na vajah. Za vsako seminarsko nalogo pripravimo po tri nadgradnje različnih težavnostnih stopenj (najlažja, srednja, najtežja), od katerih vsaka zahteva neko dopolnitev rešitve seminarske naloge. Če študent ne realizira nobene nadgradnje, vendar pa zna obrazložiti svojo rešitev seminarske naloge, prejme do 4 točke. Vsaka pravilno rešena nadgradnja prinese po dve dodatni točki. Za vsako seminarsko nalogo je tako možno zbrati največ 10 točk. Opisani način ocenjevanja seminarskih nalog nagrajuje dodatno delo, saj ima študent, ki si doma izmišljuje in rešuje različne dodatne naloge, več možnosti za večje število točk. Obenem pa takšno ocenjevanje posredno kaznuje goljufanje, saj študentje, ki naloge niso rešili samostojno, praviloma ne bodo realizirali nobene nadgradnje in pogosto ne bodo znali niti pojasniti delovanja rešitve seminarske naloge. Na dveh terminih vaj (na sredini in ob koncu semestra) študentje opravljajo približno 15-minutni test iz teorije, ki je sestavljen iz teoretičnih vprašanj in enostavnih nalog za preverjanje razumevanja ključnih konceptov. Študentje rešujejo teste na svojih računalnikih preko sistema Moodle[2], ki omogoča enostavno sestavljanje in samodejno ocenjevanje različnih tipov testov. Oba testa sta vredna po 10 točk. Priprava seminarskih nalog, nalog za nadgradnje in teoretičnih testov zahteva precejšno angažiranost pedagoškega osebja. V študijskem letu 2011/12 smo vaje izvajali v 12 skupinah. Za vsakega od treh terminov seminarskih nalog smo pripravili po 6 nalog (po eno nalogo na dve skupini), kar skupaj nanese 18 nalog. Pri nadgradnjah smo vsaki od 12 skupin namenili svoj komplet in tako skupno pripravili kar 108 tovrstnih nalog (12 kompletov s po tremi nadgradnjami za vsakega od treh terminov). Tudi pri teoretičnih testih smo za vsako skupino pripravili poseben komplet vprašanj in nalog. Za oba testa skupaj smo pripravili 24 kompletov s skupno 103 različnimi vprašanji in nalogami.

6 Motivacija boljših študentov

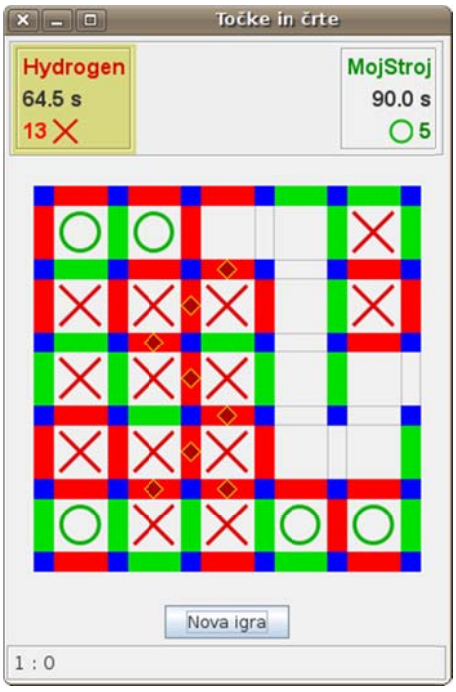
V okviru bolonjske prenove predmeta UPP-FRI smo z namenom popestritve predmeta in dodatne motivacije boljših študentov uvedli dve novosti: tekmovanja v programiranju namiznih iger in natečaje za pripravo seminarskih nalog na temo dedovanja.

6.1 Tekmovanja v programiranju namiznih iger

Tekmovanje v programiranju namiznih iger smo prvič izvedli v študijskem letu 2008/09. V letih 2008/09 in 2010/11 so študentje tekmovali v izdelavi programov za igro Štiri v vrsto, v letu 2009/10 je bila na sporedu igra Potapljanje ladjic, v letu 2011/12 pa igra Točke in črte[3] (angl. Dots and Boxes), ki jo prikazuje slika 1. Da bi se študentje lahko osredotočili zgolj na najzanimivejši del problema, torej na logiko za izbiranje potez, in da bi lahko študentski

[2] http://moodle.org/
[3] http://en.wikipedia.org/wiki/Dots_and_boxes

programi na enostaven način igrali z ljudmi in med seboj, vsakokrat pripravimo posebno grafično ogrodje. Študentje, ki želijo sodelovati na tekmovanju, morajo zgolj dopolniti ogrodje z lastno igralno logiko. Vsako tekmovanje je razdeljeno na dva dela: predtekmovanje in glavno tekmovanje. V predtekmovanju se študentski programi merijo med seboj; vsak program z vsakim odigra vnaprej določeno število iger. Prvouvrščeni program prejme 8 točk, drugouvrščeni 6, tretjeuvrščeni 4 itd. Pet najboljših programov in njihovih avtorjev se uvrsti v glavno tekmovanje, ki poteka v času predavanj. Na glavnem tekmovanju vsak program odigra po dve igri proti vsakemu od avtorjev preostalih štirih programov. Program za zmago prejme eno točko, za remi polovico točke, za poraz pa ničlo. Program, ki v skupnem seštevku (skupaj s točkami s predtekmovanja) prejme največ točk, je proglašen za zmagovalca. Njegov avtor je nagrajen z 20 dodatnimi točkami na pisnem izpitu. Ustrezno manjše število dodatnih točk prejmejo avtorji programov, ki se uvrstijo na mesta od drugega do desetega. Glavno tekmovanje je zanimiv in napet dogodek. Pri tekmovanju v igri Točke in črte, denimo, je bila zaključna razvrstitev do konca negotova. Trije šibkejši programi so z avtorji ostalih programov igrali dokaj enakovredno, najboljša dva, ki sta uporabljala številne napredne tehnike in algoritme za določanje optimalne poteze, pa sta bila prepričljivo močnejša od človeških nasprotnikov, čeprav nista zmagala v vseh igrah.



Slika 1: Najboljša študentska programa igrata igro Točke in črte med seboj.

6.2 Natečaji za pripravo seminarskih nalog iz dedovanja

Natečaj za pripravo seminarskih nalog na temo dedovanja smo prvič izvedli v študijskem letu 2010/11. Študentom, ki želijo sodelovati na natečaju, ni treba reševati tretje seminarske naloge, ki pokriva področje dedovanja, ampak namesto tega pripravijo predlog lastne seminarske naloge. Predlagana naloga mora biti pripravljena na enak način, kot so pripravljene naloge, ki jih dobijo v reševanje ostali študentje. To pomeni, da mora vsebovati besedilo in rešitev osnovne naloge (ki jo študentje rešujejo doma) ter besedila in rešitve treh različno težkih nadgradenj (ki jih morajo študentje realizirati na vajah). Pri tem morajo upoštevati, da so v sklopu naloge zajeti vsi bistveni koncepti, ki so značilni za dedovanje:

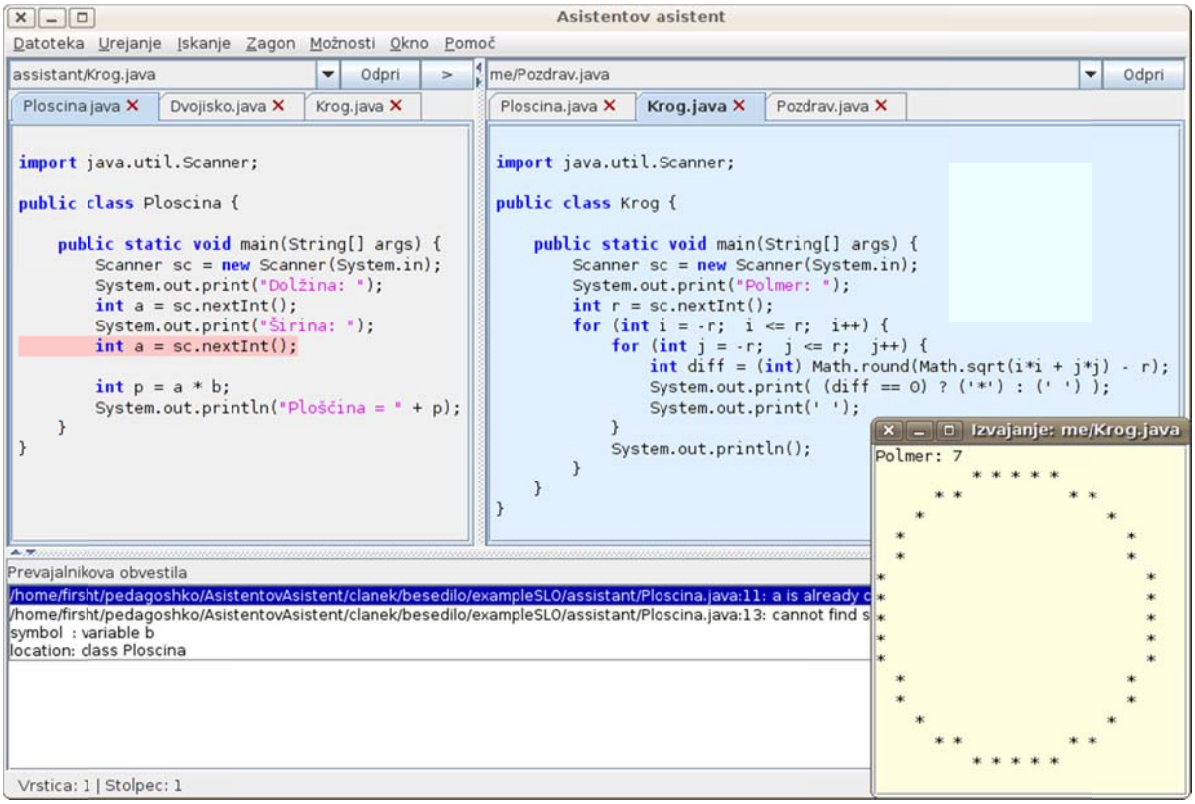
hierarhija najmanj treh razredov, deklaracije konstruktorjev, redefinicija metod in dinamično povezovanje metod. Kdor izpolni predpisane pogoje, prejme za tretjo seminarsko nalogo vseh 10 točk. Avtorji petih najboljših predlogov prejmejo še dodatne točke v okviru točkovanja sprotnega dela: najboljši 10, drugi najboljši 8, tretji 6 itd. V letu 2010/11 smo prejeli 9 predlogov, v letu 2011/12 pa 14. Tri najboljše predloge iz leta 2010/11 smo z manjšimi prilagoditvami uporabili kot seminarske naloge v letu 2011/12. Natečaji za pripravo nalog tako vzpodbujajo ustvarjalnost, obenem pa koristijo tudi pedagoškemu osebju, saj povečujejo zalogo nalog in idej zanje.

7 Programski sistem *Asistentov asistent*

Odprtokodni programski sistem Asistentov asistent4 (Fürst in Mahnič, 2012) smo razvili z namenom izboljšanja učinkovitosti izvajanja "običajnih" vaj, torej vaj, ki niso namenjene ocenjevanju seminarskih nalog. Sistem, ki smo ga prvič uvedli v študijskem letu 2010/11, opravlja več vlog, njegova najpomembnejša naloga pa je samodejni sprotni prenos datotek z asistentovega računalnika na računalnike študentov, s čimer odpravlja potrebo po pretipkovanju programske kode s platna. Študentje lahko tako zbrano sledijo asistentovi razlagi in spremljajo razvoj programa na platnu ali pa kar na svojih zaslonih, kar je še posebej uporabno v predavalnicah s slabšo preglednostjo. Samodejni prenos datotek koristi tudi asistentoma, saj se jima ni več treba ukvarjati z odpravljanjem tipkarskih napak posameznih študentov. Sistem Asistentov asistent deluje po načelu strežnik-odjemalec. Asistent na svojem računalniku požene strežnik (strežniško komponento sistema), nato pa vsak študent požene odjemalca (odjemalsko komponento). Asistent ob zagonu strežnika poda pot do imenika na svojem računalniku (denimo S), čigar vsebino želi samodejno in sproti prenašati do računalnikov študentov. Vsak študent ob zagonu odjemalca poda pot do nekega imenika D na svojem računalniku. Odjemalec najprej ustvari podimenika assistant in me znotraj imenika D, nato pa ob rednih časovnih presledkih (vsakih t sekund) posreduje strežniku podatke (imena in časovne zaznamke zadnjih sprememb) o vseh datotekah v podimeniku assistant. Strežnik nato primerja posredovane podatke s podatki o datotekah v svojem imeniku S in odjemalcu pošlje vse datoteke, ki v odjemalčevem imeniku D/assistant še ne obstajajo, in vse datoteke, ki so se v času od prejšnjega stika z odjemalcem spremenile. Poslane datoteke se prenesejo v imenik D/assistant. Na ta način bo vsak odjemalec prejel najbolj sveže različice vseh datotek iz strežnikovega imenika S najkasneje t sekund po času njihove zadnje shranjene spremembe. Za tekoč prenos mora asistent torej le dovolj pogosto shranjevati svoje datoteke ali pa preprosto vključiti možnost samodejnega shranjevanja, ki je na voljo v mnogih sodobnih urejevalnikih besedil. Poleg samodejnega prenašanja datotek sistem Asistentov asistent omogoča tudi sodelovanje med asistentom in študenti pri razvoju rešitve programerske naloge. Odjemalec deluje v okviru grafičnega okna (slika 2), razdeljenega na dve plošči. V levi plošči lahko študent pregleduje datoteke v imeniku D/assistant, v desni pa lahko pregleduje in ureja datoteke v imeniku D/me. Gumb z oznako > v sredinskem zgornjem delu okna prekopira trenutno izbrano datoteko v levi plošči (torej v imeniku D/assistant) v imenik D/me in jo obenem odpre v desni plošči. Asistent lahko tako razvije nek del programa (ta se samodejno prenaša v študentov imenik D/assistant, študent pa lahko njegovo nastajanje sproti spremlja v levi plošči), nato pa študentom naloži, naj rešitev dopolnijo. Vsak študent nato klikne na gumb z oznako > in razvije svojo dopolnitev v desni plošči. Nato svojo dopolnitev napiše še asistent, študent pa jo lahko sproti primerja s svojo. Odjemalec sistema Asistentov asistent deluje tudi kot javansko razvojno okolje, saj ponuja vse običajne sodobne možnosti

[4] http://ltpo.fri.uni-lj.si/as2/

urejanja, poleg tega pa omogoča prevajanje in izvajanje programov kar znotraj uporabniškega vmesnika. Slika 2 prikazuje rezultat prevajanja programa v levi plošči (napake so prikazane v spodnjem delu okna) in izvedbo programa v desni plošči (ločen okvirček). Sistem Asistentov asistent je znatno povečal učinkovitost izvajanja vaj. Ocenjujemo, da smo v študijskem letu 2009/10, ko se je obseg vaj že zmanjšal, sistema Asistentov asistent pa še nismo uvedli, zaradi pretipkovanja in popravljanja tipkarskih napak izgubili do 15 minut v 90-minutnem bloku vaj. Zaradi pretipkovanja študentje tudi niso mogli dovolj zbrano slediti razvoju programa. Sistem Asistentov asistent je odpravil obe težavi, poleg tega pa je omogočil kooperativno reševanje programerskih nalog. Namesto da bi študentje zgolj pasivno spremljali razvoj programa, smo v zadnjih dveh letih ubrali bistveno bolj dinamičen pristop. Asistent sedaj običajno pripravi zgolj besedilo naloge in napiše nekatere "dolgočasnejše" dele programa, kot je npr. branje podatkov s standardnega vhoda, študentje pa izdelajo preostanek. Pri obsežnejših nalogah se sodelovanje med asistentom in študenti lahko celo večkrat ponovi (npr. pri različnih metodah kompleksnejšega javanskega razreda). V času, ko študentje samostojno rešujejo trenutno zastavljeno nalogo, oba asistenta potujeta po učilnici, odgovarjata na vprašanja, pomagata z namigi itd., česar pred uvedbo sistema Asistentov asistent skoraj nismo poznali.



Slika 2: Odjemalec sistema Asistentov asistent.

8 Zaključek

V prispevku smo predstavili novosti, ki smo jih pri uvodnem programerskem predmetu na univerzitetnem študiju na Fakulteti za računalništvo in informatiko Univerze v Ljubljani uvedli kot posledico uveljavitve bolonjske reforme. Ocenjujemo, da so novosti skladne s smernicami bolonjske reforme. S spremembo načina ocenjevanja smo bistveno povečali težo in pomen sprotnega samostojnega dela. Z uvedbo tekmovanj v programiranju namiznih iger in natečajev za pripravo seminarskih nalog iz dedovanja smo popestrili predmet in dodatno

motivirali boljše študente. S pomočjo sistema Asistentov asistent smo povečali učinkovitost izvajanja vaj, katerih obseg se je po bolonjski prenovi skrčil za tretjino. V študijskem letu 2012/13 bomo poleg obveznih seminarskih nalog uvedli neobvezne tedenske domače naloge, saj bi želeli začetnike vzpodbuditi, da pričnejo programirati že takoj na začetku semestra. Zagovor prve seminarske naloge je namreč na sporedu šele četrti ali celo peti teden vaj, do tedaj pa se nabere že toliko snovi, da povprečen začetnik, ki prične programirati šele nekaj dni pred zagovorom, ne bo več zmogel ujeti priključka s tekočo snovjo. Domače naloge bi opisani problem omilile. Upamo, da bodo svoj namen dosegle kljub neobveznosti.

Viri

Anik, Z. in Baykoç, Ö.F., (2011): "Comparison of the most popular object-oriented software languages and criterions for introductory programming courses with analytic network process: a pilot study," Computer Applications in Engineering Education, let. 19, št. 1, str. 89–96.

Barak, M., Harward, J., Kocur, G. in Lerman, S., (2007): "Transforming an introductory programming course: from lectures to active learning via wireless laptops," Journal of Science Education and Technology, let. 16, št. 4, str. 325–336.

Benander, A., Benander, B. in Sang, J., (2004): "Factors related to the difficulty of learning to program in Java – an empirical study of non-novice programmers." Information and Software Technology, let. 46, št. 2, str. 99–107.

de Raadt, M., Watson, R. in Toleman, M., (2004): "Introductory programming: what's happening today and will there be any students to teach tomorrow?" Proc. 6th Conf. on Australasian Computing Education, strani 277–282, Darlinghurst, Avstralija, 2004, R. Lister in A. Young, urednika, Australian Computer Society, Inc., Avstralija.

Fernandez, C., Diez, D., Zarraonandia, T. in Torres, J., (2011): "A student-centered introductory programming course: the cost of applying Bologna principles to computer engineering education," International Journal of Engineering Education, let. 27, št. 1, str. 14–23.

Fürst, L. in Mahnič, V., (2012): "A cooperative development system for an interactive introductory programming course," World Transactions on Engineering and Technology Education, let. 10, št. 2, str. 122–127.

Hawi, N., (2010): "Causal attributions of success and failure made by undergraduate students in an introductory-level computer programming course," Computers & Education, let. 54, št. 4, str. 1127–1136.

Huet, I., Pacheco, O.R., Tavares, J. in Weir, G., (2004): "New challenges in teaching introductory programming courses: a case study," Proc. 34th Frontiers in Education Conf., zvezek 1, strani T2H/5 – T2H/9, Savannah, Georgia, ZDA, oktober 2004, IEEE.

Jenkins., T., (2002): "On the difficulty of learning to program," 3rd Annual LTSN-ICS Conf., strani 65–71, Loughborough, Združeno kraljestvo, avgust 2002, LTSN-ICS.

Mahnič, V. in Gams, M., (2003): "Some experiences in teaching introductory programming at the faculty level," World Transactions on Engineering and Technology Education, let. 2, št. 3, str. 441–444.

Mahnič., V., (2004): "Poučevanje programiranja na univerzitetnem študiju računalništva in informatike: izkušnje predavatelja in mnenja študentov," Organizacija, let. 37, št. 8, str. 507–513.

Milne, I. in Rowe, G., (2002): “Difficulties in learning and teaching programming – views of students and tutors,” Education and Information Technologies, let. 7, št. 1, str. 55–66.

Moura, I.C. in van Hattum-Janssen, N., (2011): “Teaching a CS introductory course: an active approach,” Computers & Education, let. 56, št. 2, str. 475–483.

Richards, B., (2003): “Experiences incorporating Java into the introductory sequence,” Journal of Computing in Small Colleges, let. 19, št. 2, str. 247–253.

von Glasersfeld, E., (1995): “Radical Constructivism: A Way of Knowing and Learning,” Falmer Press, London.

Wächter, B., (2004): “The Bologna process: developments and prospects,” European Journal of Education, let. 39, št. 3, str. 265–273.