PD.TVZ.HR
POLYTECHNIC & DESIGN Vol. 8, No. 4, 2020.
DOI: 10.19279/TVZ.PD.2020-8-4-05

PRIMJENA VERIFIKATORA U NASTAVI PROGRAMIRANJA ZA VRIJEME PANDEMIJE

APPLICATION OF VERIFICATOR IN TEACHING PROGRAMMING DURING A PANDEMIC

Danijel Radošević

Sveučilište u Zagrebu, Fakultet organizacije i informatike, Varaždin, Hrvatska

SAŽETAK

Verifikator je edukativno sučelje za razvoj programa u programskom jeziku C++. Prilagođeno je potrebama nastave programiranja na početnoj akademskoj razini jer zahtijeva od studenata sintaksne i logičke provjere programa uz ostale funkcije kao što su personalizacija programa, pomoć pri sintaksnim i logičkim pogreškama i kontrola vremena. Radi primjene u uvjetima online nastave, za vrijeme pandemije COVID-19, Verifikator je dorađen u dijelu koji se odnosi na praćenje rada studenata, uključujući i nedopuštene radnje, poput prekrajanja programa sa svrhom prikrivanja prepisivanja. Dosadašnja nastavna iskustva vezana uz predmet "Programiranje 2" na Fakultetu organizacije i informatike u Varaždinu pokazuju da se, zahvaljujući Verifikatoru, mogu efikasno provoditi online vježbe programiranja na kojima studenti svoje rezultate predaju na sustav Moodle, tek uz manji pad ostvarenih bodova u odnosu na kontaktnu nastavu, što se kompenzira drugim aktivnostima.

Ključne riječi: *Verifikator, C++, nastava programiranja, edukativno programsko sučelje*

ABSTRACT

Verificator is an educational interface for program development in the C ++ programming language. It is tailored to the needs of teaching programming at the initial academic level because it requires students to syntactically and logically check the program in addition to other functions such as program personalization, assistance with syntax and logic errors, and time control. For application in online teaching conditions, during the COVID-19 pandemic, the Verificator was refined in the part related to monitoring student work, including unauthorized actions, such as rewriting programs for the purpose of concealing transcripts. Previous teaching experiences related to the subject "Programming 2" at the Faculty of Organization and Informatics in Varaždin show that, thanks to the Verificator, online programming exercises can be efficiently conducted in which students submit their results to the Moodle system, tek uz manji pad ostvarenih bodova u odnosu na kontaktnu nastavu, što se kompenzira drugim aktivnostima.

Keywords: *Verificator, C++, teaching programming, educational program interface*

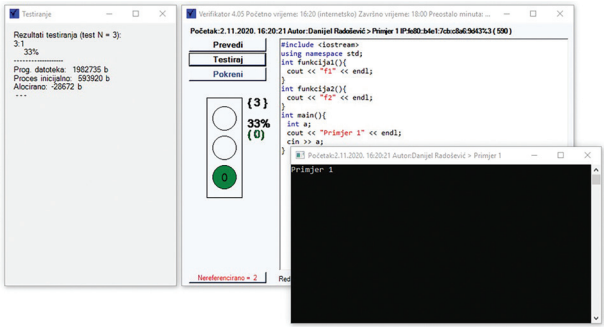
1. UVOD
1. INTRODUCTION

Početne verzije programa Verifikator uvedene su u nastavu programiranja na Fakultetu organizacije i informatike u Varaždinu još ak. god. 2008/09, kada je provedeno i prvo istraživanje o primjeni tog alata [1]. Početni zadatak bio je osigurati programsko sučelje za C++ koje će spriječiti kopiranje programskog kôda među studentima te osigurati da studenti povremeno sintaksno provjeravaju svoj programski kôd, umjesto da pišu napamet, gomilajući tako pogreške. Također, trebalo je omogućiti personalizaciju programa, tako da programski kôd u obliku komentara sadrži i određene metapodatke, uključujući prezime i ime autora, naziv programskog zadataka s kratkim opisom, početno i završno vrijeme rada na programu, te kontrolnu sumu pomoću koje se može provjeriti da program nije modificiran izvan

Verifikatora. U svom daljnjem razvoju, tokom godina, Verifikator, Slika 1, je dobivao nove alate i funkcije [2]: tutor za pomoć studentima u vezi sintaksnih i mogućih logičkih pogrešaka, alate za analizu strukture programa, mogućnost određene razine logičkog testiranja programa (provjera da program može izvršiti sve svoje programske blokove), te ograničenje vremena za rješavanje programskog zadatka.

Slika 1 *Programsko sučelje Verifikatora*

Figure 1 *Verificator program interface*



Kroz svo to vrijeme nastava se u svom praktičnom dijelu izvodila kontaktno, u računalnim učionicama. U to vrijeme jedna od ključnih funkcija Verifikatora bilo je vremensko ograničenje za izradu zadataka, kako bi se rješenja mogla provjeriti i bodovati na licu mjesta, prije dolaska sljedeće grupe studenata. Pandemija COVID-19 dovela je do prelaska nastave s kontaktnog na online način rada. Kontrola studenata postala je otežana i postavilo se pitanje vjerodostojnosti predanih studentskih rješenja, ponajprije vezano uz problem prepisivanja. Kao jedna od mjera da se prepisivanje oteža, uvedeno je da svi studenti (u slučaju predmeta "Programiranje 2", njih 239) rješavaju svoje zadatke, istovremeno, uz, naravno, više grupa zadataka. Svi izrađeni zadaci prikupe se nakon održane vježbe, te se provjeravaju na prepisivanje, najprije pomoću odgovarajućeg softvera, koji je također razvijen specifično za potrebe predmeta, a zatim i ručno, usporedbom jedan na jedan. Na prvoj tako održanoj vježbi uhvaćeno je oko 15% studenata u prepisivanju, te su kažnjeni oduzimanjem bodova s te vježbe, te mogućim strožim kaznama u slučaju ponovljenog prekršaja. Na sljedećoj vježbi broj tako uhvaćenih studenata bio je manji, oko 10%. No, pokazalo se i to da je prepisivanje postalo sofisticiranije. Pokušava se s promjenom naziva identifikatora, redoslijeda instrukcija i potprograma, promjenom stila pisanja kôda i sl. Iz tog razloga Verifikator je proširen novim mogućnostima praćenja razvoja programa. Prva je uvođenje hodograma upisa i brisanja programskog kôda, a druga praćenje korištenih identifikatora u razvoju programa. Svrha je identificiranje ponašanja gdje student najprije izradi svoje vlastito programsko rješenje, da ga drugima na prepisivanje, a svoj vlastiti program modificira na način da se nazivi identifikatora, stil i poredak pisanja instrukcija promijene, pri čemu funkcionalnost ostaje ista.

2. POVEZANI RAD
2. RELATED WORK

Do sada je razvijeno više alata sa svrhom pomoći u podučavanju programiranja. Kratak pregled aktualnih alata dat je u [3]. Dosadašnja istraživanja, poput onih koje su proveli Lane i VanLehn [4], Erol i Kurt [10] i drugi pokazala su da neki od tih alata mogu pozitivno utjecati na studentsku percepciju programiranja, te motiviranost za učenje.

Za mjerenje prihvaćenosti primjene tehnologije u području online podučavanja najpopularniji model je TAM (eng. Technology Acceptance Model) [5]. Model sugerira da kada se korisnici upoznaju s novom tehnologijom, brojni čimbenici utječu na njihovu odluku o tome kako će je i kada koristiti, a osobito *percipirana korisnost* i *percipirana jednostavnost upotrebe*. Istraživanje provedeno sa studentima 2014. godine u radu s Verifikatorom [2] ukazalo je na glavne faktore koji utječu na njegovo prihvaćanje od strane studenata. Pokazalo se da oni studenti koji su ranije pokazivali strah od programiranja, a takvih je, prema [1] oko 13%, dobro prihvaćaju Verifikator. Također, ustanovljena je i visoka korelacija između percipirane korisnosti i percipirane jednostavnosti upotreba Verifikatora.

Za sada još nema puno provedenih studija vezanih uz upotrebu pojedinih alata za podučavanje programiranja u uvjetima pandemije. Ona koja su provedena, ne bave se toliko programskim alatima, koliko nastavničkim kompetencijama i adaptaciji procesa podučavanja, kao npr. [6] i [7].

Dio istraživanja bavi se automatiziranim ocjenjivanjem u uvjetima pandemije, npr. [8], odnosno održivosti nastavnog procesa akademskog obrazovanja u vremenima pandemije [9].

Prema tome, može se uočiti da su dosadašnji radovi uglavnom orijentirani k minimiziranju štete prouzročene nastavnom procesu zbog prelaska na online nastavu. Treba međutim uočiti i određene prednosti i prilike koje donosi takav način rada. Za to su, među ostalim, potrebni i odgovarajući alati uz pomoć kojih studenti mogu svladavati određena znanja i vještine, poput onih koja su potrebna za programiranje.

3. VERIFIKATOR
3. VERIFICATOR

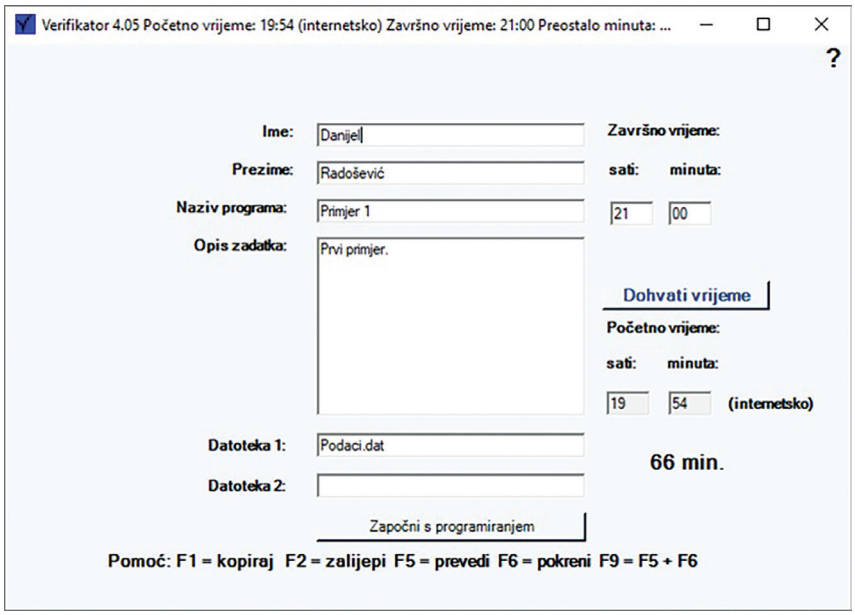
Prilikom kreiranja Verifikatora, još od njegovih početaka, pošlo se od cilja da se studentima s jedne strane pomogne u usvajanju novih znanja i vještina vezanih uz programiranje, te da ih se odvrati od nedozvoljenih aktivnosti kao što su kopiranje tuđeg kôda, programiranje napamet ili prepisivanje od kolega. Zbog toga je samo programsko sučelje (slika 1) pojednostavljeno do krajnjih granica, kako bi se korisnik mogao u potpunosti posvetiti svom programskom zadatku. U okviru tako pojednostavljenog sučelja ostvareno je nekoliko glavnih funkcija Verifikatora, kao što možemo vidjeti u sljedećem pregledu.

3.1. PERSONALIZACIJA PROGRAMA
3.1. PROGRAM PERSONALIZATION

Na početku rada s Verifikatorom studenti ispunjavaju obrazac u koji upisuju svoje prezime i ime, naziv programskog primjera, te njegov kratki opis (slika 2).

Tako uneseni podaci upisuju se, zajedno s ostalim metapodacima, u programski kôd u obliku komentara, kao u sljedećem primjeru:

```
// MD5:JTf174KKbFkJMAFgTMUyHQ==
// Verifikator 4.05
// Program:Primjer 1
// Početno vrijeme: 19:54 (internetsko)
Završno vrijeme: 21:00 Preostalo: 59 +
// Opis zadatka:Prvi primjer.
// Autor:Danijel Radošević
// Početno vrijeme:2.11.2020. 20:00:12
// Završno vrijeme:2.11.2020. 20:01:18
// Identifikatori:main,a
//   IP:fe80::b4e1:7cb:c8a6:9d43%3   (
590 )
// #:#include<iostream>,
//              Uspješnih/neuspješnih
prevodenja:1/0
#include<iostream>
using namespace std;
int main(){
      int a;
      cout << "Primjer 1" << endl;
      cin >> a;
}
```



Slika 2 *Obrazac za personalizaciju programa*

Figure 2 *Program personalization form*

Kao što vidimo u primjeru, tako dobiveni programski kôd sadrži i kontrolnu sumu (*// **MD5:***) pomoću koje se može provjeriti da nije modificiran izvan Verifikatora.

3.2. VREMENSKO OGRANIČENJE ZA RJEŠAVANJE ZADATKA
3.2. TIME LIMITATION FOR SOLVING THE ASSIGNMENT

Vrijeme za rješavanje zadatka može se ograničiti u okviru Verifikatora, na način da student upisuje završno vrijeme pisanja zadatka, što se kasnije može provjeriti iz metapodataka unutar programskog kôda. Kako bi se izbjegao problem različitog sistemskog vremena na različitim računalima, Verifikator dohvaća aktualno vrijeme s vremenskih poslužitelja na Internetu. Student u svakom trenutku može vidjeti koliko je još vremena preostalo za rješavanje zadatka. Kad vrijeme istekne, program se i dalje može pokrenuti (zadnje uspješno prevedeno stanje), ali ne i modificirati.

3.3. SPREČAVANJE KOPIRANJA PROGRAMA OD KOLEGA
3.3. PREVENTION OF COPYING PROGRAMS FROM COLLEAGUES

Verifikator sprečava kopiranje programa iz vanjskih izvora. To znači da cijeli programski kôd, s izuzetkom propisanih biblioteka, mora biti unesen ručno. Ovo svojstvo je povezano s obaveznim sintaksnim i logičkim provjerama programa unutar zadanih intervala, što otežava prepisivanje.

3.4. USVAJANJE DOBRIH PROGRAMERSKIH NAVIKA
3.4. ADOPTION OF GOOD PROGRAMMING HABITS

U okviru ranije nastavne prakse, a i provedenih istraživanja [1] pokazalo se da su studenti u određenoj mjeri skloni usvajanju loših programerskih navika kao što su učenje programskog kôda napamet, programiranje bez sintaksne i logičke provjere itd. Verifikator zahtijeva od studenata prevođenje kôda (što uključuje sintaksnu provjeru) unutar 10 novih instrukcija, što je grafički predstavljeno semaforom. Crveno svjetlo znači da nije moguće prevesti program dok se ne smanji broj novonapisanih instrukcija. Osim toga, program mora biti provjeren i logički unutar zadanih intervala, izraženih u broju programskih blokova od kojih se program sastoji. Kod takve provjere program mora proći kroz sve svoje blokove, da bi interval bio testiran sa 100%. Smisao te provjere je da se odvrati studente od pisanja programskog kôda koji ostaje nepovezan, npr. funkcija bez odgovarajućeg poziva, što se događa kod slijednog prepisivanja kôda. Postoci testiranja upisuju se u tzv. testnu datoteku, koju Verifikator kreira zajedno s glavnom programskom datotekom.

3.5. POMOĆ RADI LAKŠEG PRONALAŽENJA SINTAKSNIH I LOGIČKIH POGREŠAKA
3.5. HELP FOR EASIER DETECTION OF SYNTAX AND LOGICAL ERRORS

Osim standardnih pogrešaka (eng. Errors) i upozorenja (eng. Warnings) koje vraća prevoditelj, Verifikator za određeni broj sintaksnih i logičkih pogrešaka daje detaljnije obrazloženje, odnosno navodi mogući uzrok. Ova mogućnost je nastala na temelju nastavne prakse, jer se pokazalo da se neke pogreške kod rješavanja zadataka relativno često ponavljaju. Tu ulaze pogreške poput rasparenih vitičastih/okruglih zagrada odnosno navodnika, korištenje pogrešnog operatora, izostavljanja nekih često korištenih biblioteka i sl. Verifikator također uključuje i alate za analizu strukture programa, te korištenje prekidnih točaka u programu (eng. Breakpoints).

4. UPOTREBA VERIFIKATORA U KONTAKTNOJ I ONLINE NASTAVI
4. USE OF VERIFICATOR IN CONTACT AND ONLINE TEACHING

U odnosu na kontaktnu nastavu, gdje je studente moguće vizualno kontrolirati radi sprečavanja

prepisivanja, za potrebe online nastave, način rada morao je biti promijenjen. Umjesto više nastavnih grupa koje rješavaju svoje zadatke u računalnim učionicama, svi studenti na predmetu rješavaju svoje zadatke istovremeno, a povezani su putem Interneta. Na kraju vježbe treba predati na sustav Moodle dvije datoteke: programsku datoteku i tzv. testnu datoteku, koja sadrži i informaciju o provedenim testiranjima. Pri obradi rezultata provjerava se kontrolna suma svih programskih datoteka. Odgovarajući alat ujedno ispisuje i podatak o testiranim programskim intervalima (slika 3).

U sljedećem koraku datoteke se automatiziranim postupkom uspoređuju, radi traženja mogućih prepisivanja, što rezultira listom kandidata, kao u sljedećem primjeru:

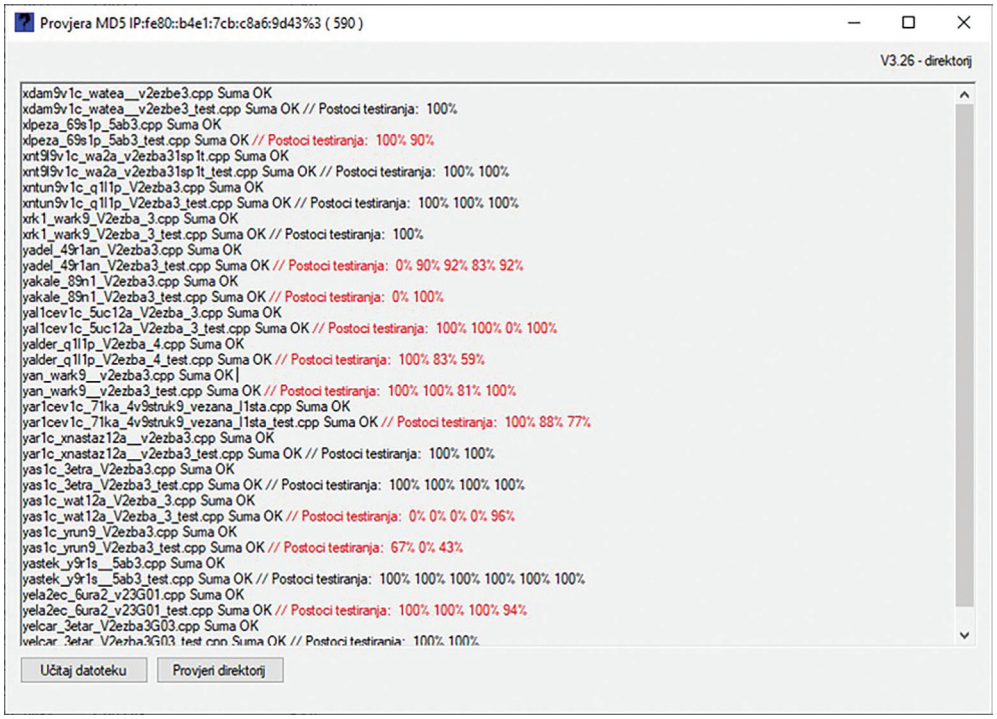
```
039% yar1c_xnastaz12a__v2ezba3.cpp : yakale_89n1_V2ezba3.cpp 123 48
038% yakale_89n1_V2ezba3.cpp : yar1c_xnastaz12a__v2ezba3.cpp 125 48
035%     yan_wark9__v2ezba3.cpp     : yela2ec_6ura2_v23G01.cpp 194 69
034%    yas1c_wat12a_V2ezba_3.cpp    : xdam9v1c_watea__v2ezbe3.cpp 91 31
034% yar1cev1c_71ka_4v9struk9_vezana_l1sta.cpp : yar1c_xnastaz12a__v2ezba3.cpp 128 44
034% yar1cev1c_71ka_4v9struk9_vezana_l1sta.cpp  :  yakale_89n1_V2ezba3.cpp 126 44
034% yar1cev1c_71ka_4v9struk9_vezana_l1sta.cpp  :  xdam9v1c_watea__v2ezbe3.cpp 83 29
033%    yela2ec_6ura2_v23G01.cpp    : yakale_89n1_V2ezba3.cpp 133 44
033%    yas1c_wat12a_V2ezba_3.cpp    : yar1cev1c_71ka_4v9struk9_vezana_l1sta.cpp 149 50
. . .
```

Datoteke se nakon toga moraju ručno usporediti. Kod toga mogu pomoći metapodaci koje Verifikator upisuje u programsku i testnu datoteku.

4.1. PRAĆENJE KORIŠTENIH IDENTIFIKATORA U PROGRAMU

4.1. TRACKING OF USED IDENTIFIERS IN THE PROGRAM

Prepisivanje obično funkcionira tako da postoji jedan autor programskog rješenja i jedan ili više onih koji prepisuju. Kako bi se prikrili tragovi prepisivanja, autor mora brzo izraditi zadovoljavajuće funkcionalno rješenje zadatka, dati ga drugima da ga prepišu, nakon čega svi sudionici nastoje prikriti tragove prepisivanja.



Slika 3 *Provjera kontrolnih suma*

Figure 3 *Checking checksums*

Kako je pokazala dosadašnja praksa, to se uglavnom radi nizom transformacija u programu, nakon kojih funkcionalnost ostaje ista, ali se program učini teško usporedivim sa svojim originalom, odnosno kopijom. Obično u tome sudjeluju i autori i prepisivači, s tim da su autori obično kreativniji, odnosno, sposobni u znatno većoj mjeri transformirati svoj program, a da zadrži funkcionalnost. Provedene transformacije u programu odnose se dijelom na promjenu redoslijeda instrukcija i potprograma, te promjenu stila pisanja programa (npr. jedna ili više instrukcija u jednom redu, uvlake i sl.) kako bi se otežala usporedba, ali i na promjenu naziva identifikatora poput naziva varijabli, struktura i potprograma. Iz tog razloga je za potrebe online nastave u Verifikatoru uvedena mogućnost praćenja korištenih identifikatora u programu. U tu svrhu, metapodaci unutar programskog kôda su prošireni podacima o korištenim identifikatorima. Sljedeći primjer to pokazuje:

```
#include<iostream>
using namespace std;
struct tstudent{
      int mat_br;
      char prez_ime[40];
};

void funkcija1(int argument1){
      cout << "argument1 = " <<
argument1 << endl;
};
int main(){
      int a;
      tstudent s1,*s2;
      funkcija1(a);
      cin >> a;
}
```

Rezultirajući niz identifikatora zapisuje se ovako:

```
//            Identifikatori:mat_br,
prez_ime[40],funkcija1,argument1,mai
n,a,s1,*s2
```

Ako se u daljnjem razvoju programa nazivi identifikatora promijene, ostaju stari i novi nazivi:

```
//            Identifikatori:mat_br,
prez_ime[40],funkcija1,argument1,mai
n,a,s1,*s2,funkcija1b,argument1b
(u primjeru su promijenjeni naziv
funkcije i naziv argumenta)
```

Prema tome, kod usporedbe programskih datoteka 1:1, može se utvrditi postojanje istoimenih identifikatora, te njihova naknadna promjena.

4.2. HODOGRAM AKTIVNOSTI U RAZVOJU PROGRAMA

4.2. TIMETABLE OF PROGRAM DEVELOPMENT ACTIVITIES

Hodogrami aktivnosti odnose se na aktivnosti upisa, odnosno utipkavanja programskog kôda, te se upisuju tzv. testnu datoteku, kao u sljedećem primjeru:

```
// H1:120100000002310
// H2:766500000003121
```

Oznaka ***H1*** odnosi se na brisanje kôda pomoću tipki ***Delete***, odnosno ***BackSpace***, dok se ***H2*** odnosi na utipkavanje znakova, odnosno pisanje programskog kôda. Svaki znak u hodogramu odnosi se na vremenski interval od jedne minute, pri čemu je skala eksponencijalna (0: nema aktivnosti, 1: jedan unos/brisanje, 2: 3 do 4 unosa/brisanja itd.). Na taj način može se pratiti dinamika izrade programa.

4.3. POSTIGNUTI REZULTATI STUDENATA U ODNOSU NA KONTAKTNI OBLIK NASTAVE

4.3. ACHIEVED RESULTS OF STUDENTS IN RELATION TO THE CONTACT FORM OF TEACHING

Praćeni su rezultati studenata na vježbama iz predmeta "Programiranje 2", i to prve četiri vježbe ak. god. 2019/20 i prve četiri vježbe u tri prethodne akademske godine. U prve tri godine nastava se provodila kontaktno, a na zadnjoj online. Broj mogućih bodova koje studenti mogu ostvariti iznosio je 0-2 na uvodnoj vježbi, te 0-4 na daljnjim vježbama. Uvodna vježba sastojala se samo od unosa zadanih programskih primjera pomoću Verifikatora (radi prilagodbe na sučelje) dok su na daljnjim vježbama studenti rješavali zadatke za koje su unaprijed dobili samo model. Broj bodova koje su studenti ostvarili na početne četiri vježbe bio je sljedeći (tablica 1):

Ak. god.	Uvodna vježba (2b)	Vježba 1 (4b)	Vježba 2 (4b)	Vježba 3 (4b)	Broj studenata
2017/18	1,98	3,25	3,27	2,63	146
2018/19	1,95	3,29	3,18	2,51	272
2019/20	2,00	3,55	3,41	2,39	160
Prosjek kontaktno	**1,98**	**3,36**	**3,29**	**2,51**	**193**
2020/21 (online)	1,77	2,97	2,52	1,53	239
Razlika online-kontaktno (%)	10,46 %	11,69 %	23,33 %	39,04 %	

Tablica 1. *Ostvareni bodovi na vježbama*

Table 1. *Achieved points on exercises*

Kao što možemo vidjeti u tablici (tablica 1), broj ostvarenih bodova studenata nešto je manji kod online nastave u odnosu na kontaktnu, ali taj pad uglavnom nije drastičan. Radi kompenzacije, studentima je uvedena mogućnost sudjelovanja sa svojim programskim primjerima u forumu, što se boduje kao dodatna aktivnost..

5. ZAKLJUČAK
5. CONCLUSION

Organizacija nastave programiranja u doba pandemije zahtjevan je zadatak. Rješavanje programskih zadataka kod kuće dovodi do ozbiljne opasnosti od nedozvoljenih radnji, poput prepisivanja. Kod toga se ne možemo u potpunosti osloniti samo na mogućnosti glasovnog i slikovnog kontakta sa studentima putem mrežnih servisa. Iz tog razloga smo, u okviru nastave predmeta "Programiranje 2" na Fakultetu organizacije i informatike u Varaždinu, odlučili prilagoditi naše edukativno programersko sučelje za C++, Verifikator, novom režimu rada. Naglasak je stavljen na praćenje rada studenata kako bi se ušlo u trag nedozvoljenim radnjama, poput prikrivanja prepisivanja naknadnim modificiranjem programa. U konačnici, došlo je do manjeg pada broja ostvarenih bodova studenata u odnosu na prethodne godine, ali je taj pad kompenziran drugim aktivnostima na kojima studenti također mogu ostvariti bodove.

Ono što je prednost sadašnjeg online načina rada jest bolja dokumentacija svih studentskih aktivnosti tokom semestra u odnosu na dosadašnji kontaktni oblik nastave, budući da sada svi riješeni zadaci ostaju pohranjeni na sustavu Moodle i moguće ih je po potrebi ponovo analizirati, u slučaju sumnje na nedopuštene radnje.

6. REFERENCE
6. REFERENCES

[1.] Radošević, D., Orehovački, T., Lovrenčić, A: "Verificator: Educational Tool for Learning Programming", Informatics in Education, Vol. 8, No. 2, pg. 261-280, ISSN 1648-5831, Institute of Mathematics and Informatics, Vilnius, Lithuania, 2009.

[2.] Orehovački, T., Radošević, D., Konecki, M.: "Acceptance of Verificator by Information Science Students", 34th International Conference on Information Technology Interfaces (ITI 2012), ISBN 978-953-7138-24-0, Cavtat, 25.-28. June 2012.

[3.] Chakraverty, S., & Chakraborty, P. "Tools and Techniques for Teaching Computer Programming: A Review", Journal of Educational Technology Systems, 49(2), 170–198. (2020) https://doi.org/10.1177/0047239520926971

[4.] H. C. Lane and K. VanLehn, "Teaching the tacit knowledge of programming to novices with natural language tutoring", Computer Science Education, vol. 15, no. 3, pp. 183-201, 2005., DOI: 10.1080/08993400500224286

[5.] S. H. Chang, C. H. Chou and J. M. Yang, "The Literature Review of Technology Acceptance Model: A Study of the Bibliometric Distributions", PACIS Proceedings, 2010., ISBN: 978-1-7336325-3-9

[6.] König, J., Jäger-Biela D. J. & Glutsch N. "Adapting to online teaching during COVID-19 school closure: teacher education and teacher competence effects among early career teachers in Germany", European Journal of Teacher Education, 43:4, 608-622, (2020) DOI: 10.1080/02619768.2020.1809650

[7.] Rapanta, C., Botturi, L., Goodyear, P. et al., "Online University Teaching During and After the Covid-19 Crisis: Refocusing Teacher Presence and Learning Activity". Postdigit Sci Educ 2, 923–945 (2020). https://doi.org/10.1007/s42438-020-00155-y

[8.] Barra, E.; López-Pernas, S.; Alonso, Á.; Sánchez-Rada, J.F.; Gordillo, A.; Quemada, J. Automated Assessment in Programming Courses: A Case Study during the COVID-19 Era. Sustainability 2020, 12, 7451., DOI: 10.3390/su12187451

[9.] C. Lecon, "Corona E-Learning Cocktail : Sustainability of University Education in Times of Pandemics," 2020 15th International Conference on Computer Science & Education (ICCSE), Delft, Netherlands, 2020, pp. 57-65, doi: 10.1109/ICCSE49874.2020.9201619.

[10.] Erol, O., Kurt, A.A. "The effects of teaching programming with Scratch on pre-service information technology teachers' motivation and achievement", Computers in Human Behavior, 77 (2017), pp. 11-18, https://doi.org/10.1016/j.chb.2017.08.017

AUTOR · *AUTHOR*

● Danijel Radošević
Rođen je 27.03.1969. u Zagrebu. Diplomirao je na Fakultetu organizacije i informatike u Varaždinu gdje sada predaje na više predmeta vezanih uz programiranje. Voditelj je Laboratorija za generativno programiranje i strojno učenje. Područja znanstvenog interesa su mu automatsko programiranje, obrada prirodnog teksta i podučavanje u programiranju.

Korespondencija · *Correspondence*
danijel.radosevic@foi.hr