UDK: 004.4:378.1 Stručni rad

SOFTVERSKI ALAT ZA INTERAKTIVNU SAMOEVALUACIJU PROGRAMSKIH ZADATAKA U VISOKOŠKOLSKOM OBRAZOVANJU

A SOFTWARE TOOL FOR INTERACTIVE SELF-EVALUATION OF PROGRAMMING ASSIGNMENTS IN HIGHER EDUCATION

Petar Marić[1], Srđan Popov[1], Rade Radišić[1], Tamara Komnenić[1]
[1]*Fakultet tehničkih nauka, Univerzitet u Novom Sadu*
{petarmaric, srdjanpopov, radisic.rade, komnenic.tamara}@uns.ac.rs

Apstrakt: *Efikasno savladavanje i primena osnovnih principa programiranja zahteva od studenata prve godine osnovnih akademskih studija da usvoje analitički pristup rešavanju problema, nezavisno od same prirode programskog jezika korišćenog u nastavi. U toj fazi školovanja uobičajeno je da proces sistematične provere validnosti rešenja prevazilazi tehničko predznanje studenata, što može stvoriti pojavu privida tačnog rešenja usled zapostavljanja karakterističnih test slučajeva i ograničenja problema. U radu je predstavljen softverski alat koji služi za interaktivno praćenje i proveru razvoja programskog zadatka. Takvim pristupom student brže dolazi do povratne informacije da li je na dobrom putu da na tačan način reši zadati problem. Upotrebom opisanog alata student u suštini izvršava samoevaluaciju svog rešenja, čime i nastavnik istovremeno dobija objektivne okvire na koje može da se osloni prilikom praćenja i ocene uspešnosti studenta u savladavanju kursom predviđenog gradiva.*
Ključne reči: *procesom vođena samoevaluacija, softversko inženjerstvo, testiranjem-vođen razvoj softvera*

Abstract: *Mastering of basic principles of programming and its usage requires from first year students of Bachelor studies to adopt analytical approach for problem solving, regardless of programming languages' nature used in teaching. In that learning phase it is common that systematic validation process of solution exceeds student's technical foreknowledge, which tends to create the appearance of correct solution due to neglecting of characteristic test cases and problem constraints. In this paper, a programming tool is presented, which can be used for interactive tracking and checking of programming assignment's development progress. With approach like that, student gets faster feedback about whether he is on the right track to correctly solve given problem. Usage of such tool gives student a way to self-evaluate his problem solution, by which the teacher simultaneously receives an objective framework for tracking progress*

and grading student's success in outperforming material covered by programming course.
Key words: *process driven self-evaluation, software engineering, test-driven development*

1. Uvod

Primena Bolonjske deklaracije u Srbiji uspešno je podigla nivo interakcije između studenata i nastavnog osoblja. Tokom semestra od studenata se očekuje da aktivno prate i učestvuju u nastavi, usvajaju novo gradivo i predstavljaju usvojeno kada se to od njih zahteva, kroz određene vidove provera znanja. Sa druge strane, nastavnik redovno vodi evidenciju rada i uspešnosti studenata na osnovu kojih donosi zaključak o njihovim konačnim ocenama.

Studenti na prvoj godini osnovnih akademskih studija (OAS) prolaze kroz izvestan period privikavanja na način studiranja predviđen Bolonjskom deklaracijom. U slučaju osnovnih akademskih kurseva na temu programiranja i programskih jezika, studenti se susreću sa za njih novom metodom rada, koja podrazumeva aktivnu upotrebu računara u rešavanju konkretnih problema, što se razlikuje od tradicionalnog načina usvajanja, ponavljanja i uvežbavanja gradiva na koje su se navikli tokom svog prethodnog školovanja.

2. Opis problema

Efikasno savladavanje i primena osnovnih principa programiranja zahtevaju od studenata da usvoje novi proces rešavanja problema, bez obzira na prirodu programskog jezika korišćenog u nastavi. U opštem slučaju takav proces rada je iterativne prirode, pošto se putem pokušaja i grešaka iznova primenjuju dolenavedeni koraci, sve dok se ne dobije softversko rešenje zadovoljavajućih karakteristika:

1. pisanje programskog koda
2. transformacija programskog koda u mašinski oblik
3. izvršavanje programa na računaru
4. provera validnosti softverskog rešenja i njegovih rezultata

Uobičajeno je da tehničko predznanje potrebno za sistematičnu proveru validnosti rešenja prevazilazi znanje studenata na početku prve godine osnovnih akademskih studija i samim tim im otvara mogućnost pojave *privida* tačnog rešenja problema, jer su određena ograničenja i test slučajevi zapostavljeni usled nedovoljne tehničke kompetencije i iskustva studenata. Slični problemi prisutni su i u inženjerskoj praksi, tj. u opštem slučaju ne može se uspostaviti direktan odnos između nivoa tehničke kompetencije i stepena korektnosti rešenja, usled toga što tokom izrade softvera obično nisu unapred poznata sva ograničenja i specijalni slučajevi.

Usled uvođenja novih funkcionalnosti i poboljšavanja postojećih karakteristika, kompleksnost softverskog rešenja vremenom će rasti; taj rast je obično praćen različitim vidovima testiranja, kako manuelnim tako i automatizovanim, zarad smanjenja rizika uvođenja novih softverskih defekata – odnosno zarad omogućavanja nezavisnosti tekuće verzije rešenja od novouvedenih izmena.

Jedan od ciljeva automatizacije procesa testiranja je i ravnomernija raspodela napora koji je potrebno uložiti u proveru validnosti softverskog rešenja tokom njegovog razvoja. U slučaju uspešne automatizacije procesa testiranja, moguće je na vreme primetiti da li bi

neka izmena uticala na validnost nekog od starijih delova rešenja. Takve izmene suštinski unose nove defekte u softversko rešenje te stoga ne smeju propagirati ka konačnoj verziji rešenja, već ih je potrebno što pre vratiti na dodatne ispravke i ponavljati automatizovani proces provere do uklanjanja uočenih defekata.

Moderni pristupi u razvoju softvera, kao što su *test-driven development* [1] i upotreba *Continuous Integration* sistema [2], preporučuju takvu strategiju rešavanja problema, ali se nažalost ovako disciplinovan pristup u praksi ne može očekivati od studenata koji se po prvi put susreću sa programiranjem kao disciplinom, već je potreban niz godina [3] da bi takva praksa zaživela. Ipak, korisno bi bilo izložiti studente što je pre moguće osnovnim principima dobre prakse razvoja softvera na njima smislenim primerima, da bi se izbeglo uspostavljanje loših navika i kako bi se što pre osposobili da samostalno razvijaju profesionalna softverska rešenja.

3. Predlog rešenja

Približavanje dobrih softverskih praksi za provere validnosti rešenja treba biti jednostavno za studenta, kako za razumevanje, tako i za upotrebu. Jedan od pristupa može biti i upotreba alata koji pomaže studentu da što efikasnije i jednostavnije interaktivno proverava svoje rešenje.

Postavlja se pitanje na koji način je potrebno pristupiti problemu da bi student osetio dobrobit alata dok ga koristi, kako bi, zbog svoje jednostavnosti, u što kraćem roku uvideo njegove prednosti i eventualno ga usvojio u svom daljem radu na predmetu.

Obimi početničkih zadataka iz osnova programiranja su skromni i podrazumevaju pravljenje konzolnih aplikacija sa određenim operacijama unosa i ispisa podataka. Te operacije mogu biti vršene tako što program očekuje unos sa tastature i ispisuje dobijeni rezultat na ekran uređaja. U narednoj fazi učenja programiranja, uključuje se i mogućnost rada unosa i ispisa podataka sa datotekama umesto tastature i ekrana.

Sa druge strane, podrazumeva se uvođenje mehanizama koji podržavaju modularnost i ponovnu iskoristivost u kodu putem upotrebe koncepta programskih procedura i funkcija (kao osnovnih jedinica programa). U tom slučaju, koncept jediničnog testiranja [1] predstavlja rešenje koje je sastavni deo dobre prakse u razvoju kvalitetnog softvera. Direktna izloženost studenata jediničnom načinu testiranja bila bi, u ovom slučaju, kontraproduktivna iz više razloga:

- pisanje procedura i funkcija zbog svojih karakteristika ne mogu biti sastavni delovi prvih lekcija uvoda u programiranje, već se uvode naknadno i postepeno
- jedinični testovi podrazumevaju pisanje programskog koda koji će biti zadužen za proveru korektnosti osnovne jedinice programa
- sloboda programskog rešenja bila bi sužena ukoliko bi nastavnik određivao i zadavao jedinične testove na osnovu kojih se očekuje programsko rešenje

Iz navedenih razloga, korisnost alata koji se direktno oslanja na koncept jediničnog testiranja bio bi preobiman za osobu koja je u fazi ovladavanja osnovnim veštinama programiranja. Optimalno rešenje bilo bi takvo da alat može na određeni način da

apstrahuje proces pisanja jediničnih testova tako da ne umanjuje mogućnost efikasne i temeljne validacije programskog rešenja. U slučaju programskih rešenja manjeg obima to je izvodljivo na način da se čitavo izvršivo programsko rešenje posmatra kao jedinica koja će se testirati. Rešenje treba korektno da funkcioniše i u slučaju različitih izvora ulazno-izlaznih podataka, zbog ugrađene apstrakcije preuzete iz *Unix* filozofije, o datoteci kao primitivi ulazno-izlaznih operacija.

Metoda provere rešenja podrazumevala bi automatizovano izvršavanje serije test scenarija po principu crne kutije, gde bi se za određeni set ulaznih vrednosti, validacija vršila na osnovu unapred očekivanih izlaznih vrednosti. Navedeni skupovi ulaznih/izlaznih vrednosti zapisani su na razumljiv način da bi student mogao i manuelno da testira svoje rešenje. Kriva obuke u korišćenju alata ima strmu uzlaznu putanju, a tako je podešena da bi student mogao što brže da se privikne na proces automatizovane provere validnosti programskog rešenja.

Struktura studentskog zadatka tada bi se sastojala iz sledećih elemenata:

- tekst zadatka
- datoteka u kojoj se očekuje da student piše izvorni kod rešenja problema
- *Python* [4] bazirani alat koji pokreće provere validnosti rešenja
- direktorijum koji sadrži sve test slučajeve kojima se ispituje programsko rešenje

Ovakvim pristupom student može postepeno da razvija svoje programsko rešenje, pri čemu u svakom trenutku može da proveri:

- sintaksnu korektnost programa, koja je deo procesa transformacije u mašinski oblik programa
- korektnost rezultata izvršavanja programa, koja podrazumeva pokretanje programa za svaki slučaj testiranja posebno

Test slučajevi nalaze se u posebnom, za njih predviđenom, direktorijumu. Oblik pojedinačnog test slučaja je uvezani par datoteka koje sadrže, respektivno, ulazne vrednosti i očekivani izlazni oblik, koji je programsko rešenje dužno da ispoštuje. Inicijalne test slučajeve bi zadavao nastavnik, kao kompetentnije lice koje na osnovu svog iskustva ima povratnu informaciju o čestim greškama i karakterističnim nedostacima u studentskim rešenjima. Datoteke koje opisuju test slučajeve su namenski u jednostavnom tekstualnom formatu, da bi se studentima olakšalo dodavanje sopstvenih test slučajeva.

4. Studija slučaja

Način rada opisanog alata za proveru rešenja prikazaće se na zadatku datom studentima prve godine studijskog programa OAS „Energetika, elektronika i telekomunikacije“ u redovnom terminu računarskih vežbi na predmetu „Programski jezici i strukture podataka“ koje se izvode na Fakultetu tehničkih nauka, Univerziteta u Novom Sadu:

```
Dat je niz A od maksimalno 20 realnih elemenata.
Učitati n elemenata, a zatim naći maksimalnu vrednost niza.

Za sledeće ulazne podatke:
```

```
        n = 5
        A = {4, 2, 6, -7, 1}

očekivani izlaz je u sledećem formatu:

        A[0] = 4.00
        A[1] = 2.00
        A[2] = 6.00
        A[3] = -7.00
        A[4] = 1.00

        max = 6.00


Savetuje se upotreba alata za automatizovanu proveru kvaliteta
Vašeg rešenja:

    ./proveri_resenje --help
    ./proveri_resenje moje_resenje.c
```

U pitanju je relativno jednostavan zadatak koji bi trebalo rešiti tokom trajanja termina računarskih vežbi na kojima se izučavaju osnove rada sa nizovima i matricama u programskom jeziku C [5].

Jedan od studenata došao je do sledećeg rešenja, koje prolazi proces kompajliranja:

```
#include <stdio.h>

#define MAX_SIZE 20

int main() {
    double A[MAX_SIZE], max;
    int i, n;

    do {
        printf("Unesite broj clanova niza: ");
        scanf("%d", &n);
    } while(n <= 0 || n > MAX_SIZE);

    for(i = 0;i < n;i++) {
        printf("A[%d] = ", i);
        scanf("%lf", &A[i]);
    }

    for(i = 0;i < n;i++) {
        printf("A[%d] = %.2lf\n", i, A[i]);
    }

    max = 0;
    for(i = 0;i < n;i++) {
        if(A[i] > max) {
            max = A[i];
        }
    }

    printf("max = %.2lf\n", max);

    return 0;
}
```

Međutim, asistent koji je pregledao navedeni programski kod uočio je jednu logičku grešku (naznačenu u gornjem listingu) u određivanju maksimalne vrednosti niza, koja može dovesti do netačnih rezultata za pojedine ulaze.
Studentu je predočeno da njegov programski kod poseduje izvesne nedostatke i predložena mu je upotreba alata za automatizovanu proveru rešenja.

U ovom radu je, zarad bolje preglednosti ispisa alata, broj korišćenih test scenarija smanjen je na dva osnovna: prvi, pomoću kog se proverava osnovna funkcionalnost rešenja, sa pozitivnim elemantima niza i drugi, pomoću kog se proverava ponašanje rešenja kada su svi elementi u nizu isključivo negativni brojevi. Rezultujući ispis alata na osnovu prethodno navedenih slučajeva je sledeći:

```
[INFO] PASSED: test case 'normal:example'
[WARNING] UNSURE test case 'normal:negatives': Unsure if test
cases passed, please check:
        Expected:
            A[0] = -4.00
            A[1] = -2.00
            A[2] = -6.00
            A[3] = -7.00
            A[4] = -1.00

            max = -1.00
        Got:
            Unesite broj clanova niza: A[0] = A[1] = A[2] = A[3] =
A[4] =
            A[0] = -4.00
            A[1] = -2.00
            A[2] = -6.00
            A[3] = -7.00
            A[4] = -1.00
            max = 0.00
```

Iz rezultujućeg ispisa alata vidi se da je studentovo rešenje prošlo test scenario „normal:example“, što predstavlja prvi slučaj, dok je za drugi test scenario „normal:negatives“ alat izbacio upozorenje sa očekivanim ispisom i ispis dobijen od strane studentovog rešenja. Primetno je da se ispisi razlikuju u vrednosti maksimalnog elementa, odakle se može videti da se rešenje koje je student priložio ne izvršava korektno za slučaj kada su elementi niza isključivo negativni brojevi. U pitanju je veoma česta logička greška u postavljanju inicijalne vrednosti maksimuma, pošto studenti prilikom manuelnog testiranja uglavnom isprobavaju isključivo test slučaj koji je opisan tekstom zadatka (tj. „normal:example“ u notaciji alata).

Student je na osnovu ispisa alata i dodatnog manuelnog testiranja uspeo da izoluje deo programskog koda koji je smatrao da dovodi do netačnih rezultata i potom, nakon kraće diskusije sa svojim asistentom, da formuliše ispravno rešenje (`max = A[0];`).

5. Diskusija

Opisani alat omogućava studentu da veoma brzo dobije povratnu informaciju da li je na dobrom putu da reši problem koji je postavljen pred njega. Student ima priliku da na relativno jednostavan način usvaja i proširuje svoju svest o dobrim praksama u razvoju

softvera, što za posledicu utiče na studenta da češće proverava svoj kod, brže uočava i ispravlja greške.

Mogućnost testiranja softvera daje studentu i slobodu izmene rešenja u cilju dobijanja efikasnijeg i čitljivijeg koda, jer se temeljnim i pravovremenim testiranjem softvera može predupredi mogućnost da neka od novih izmena pokvari postojeće rešenje. Ukoliko do toga ipak i dođe, student će imati znak da u nekom trenutku nije ispoštovao proces i da bi trebalo da se vrati na neku od prethodnih verzija programskog rešenja koja su uspešno prolazila testove.

Nastavnici takođe imaju višestruke koristi od ovakvog alata za testiranje programskog koda studenata. Osim bržeg uočavanja koje karakteristične test slučajeve programski kod studenata ne zadovoljava, moguće je iskoristiti alat u laboratoriji gde se održavaju računarske vežbe kako bi se aktivno pratili podaci i pravili izveštaji o:

- prisutnosti i aktivnosti studenata na računarskim vežbama
- nivou tehničke kompetencije studenata i njihove sposobnosti da se izbore sa novim problemima

Na primer, od studenata se može tražiti da tokom trajanja računarskih vežbi ostavljaju svoja rešenja na unapred predviđeno mesto, na osnovu kojih bi nastavnik dobijao goreopisane izveštaje. Izveštaji bi se automatizovano formirali na osnovu provere studentskih rešenja u odnosu na test slučajeve koje su studenti imali na uvid, ali bi nastavnik imao mogućnost da uvede i dodatne („skrivene“) test slučajeve ukoliko to smatra potrebnim. Takvi izveštaji se onda mogu koristiti kao povratna sprega student/nastavnik, na osnovu koje se mogu donositi odluke sa ciljem boljeg iskustva studenata na predmetu i njihovog efikasnijeg savladavanja gradiva.

Usled korišćenja Python programskog jezika sam alat je, sa praktične strane, platformski neutralan [6]. Stoga, opseg upotrebe alata nije ograničen isključivo na laboratoriju gde se održavaju računarske vežbe, već je moguće isti princip rada primeniti i na probleme koje studenti mogu dobiti u formi zadatka za samostalno vežbanje kod kuće. U tom slučaju, student ima više vremena da detaljno izuči problem, tako da nastavnik može namerno da izostavi pojedine karakteristične test slučajeve sa ciljem da ih student samostalno otkrije, čime studenti dodatno razvijaju svoje analitičke sposobnosti.

Opisani alat takođe nije ograničen samo na proveru rešenja pisanih u programskom jeziku C, već može podržati sve programske jezike za koje je GCC [7] programski prevodilac sposoban da napravi izvršnu datoteku. Uz to alat je dovoljno fleksibilan da, po potrebi, može automatski da razreši neke osnovne razlike u formatiranju ispisa rezultata (velika/mala slova, drugačije formatirani ispis teksta i realnih brojeva, itd).

Od 2016/2017 školske godine alat za samoevaluaciju je u eksperimentalnoj primeni nad ograničenim brojem studentskih grupa, sa ciljem da se u narednom periodu predloženi način rada i ovaj alat uvedu u punu primenu. Softverski alat sličnih karakteristika se već godinama uspešno primenjuje na Fakultetu tehničkih nauka Univerziteta u Novom Sadu, kao ispomoć asistentima prilikom pregledanja programskih rešenja sa kolokvijuma, na više kurseva koji pokrivaju osnove programiranja.

6. Zaključak

Uvođenjem alata za proveru validnosti rešenja kao nastavnog sredstva, podiže se nivo discipline, kako studenta tako i nastavnika, u rešavanju problema odnosno evaluaciji rešenja. Student dobija priliku da kroz razne test scenarije proširi svest o ograničenjima algoritama, kako bi stekao naviku o efikasnoj obradi graničnih vrednosti za algoritme koje će naučiti u okviru kurseva koji se bave osnovama programiranja, nezavisno od konkretnog programskog jezika. Pomoću alata u svakom trenutku može izmeriti svoj progres, samim tim vršeći proces samovrednovanja sopstvenog rada, dok nastavnik, između ostalog, dobija određene objektivne okvire na koje će moći da se osloni prilikom ocenjivanja studentskog rešenja i progresa generalno.

U slučaju perzistencije podataka dobijenih na osnovu ocenjivanja studentskih rešenja na ovakav način, moguće je predstaviti rezultat čitave generacije studenata u formi statističkih podataka, dok se u slučaju proširenja opsega na čitav niz generacija dobija verodostojnija povratna informacija o generalnoj uspešnosti savladavanja gradiva. Takve informacije mogu pomoći i u analizi mogućih pravaca poboljšavanja nastavnog plana i programa, sa ciljem efikasnijeg savladavanja gradiva od strane studenata.

Literatura

[1] Python Software Foundation, Python 2.7.13 documentation: unittest – unit testing framework, 2017. [Online]. Available: https://docs.python.org/2/library/unittest.html

[2] Jenkins CI community, Jenkins CI - An extendable open source continuous integration server, 2017. [Online]. Available: http://jenkins-ci.org/

[3] P. Norvig, “Teach Yourself Programming in Ten Years“, 2014. [Online]. Available: http://norvig.com/21-days.html

[4] G. Van Rossum, “Python programming language”, 1994

[5] B. W. Kemighan and D. M. Ritchie, The C programming language, 1st ed. Prentice Hall, 1978.

[6] Cunningham & Cunningham, Inc., “c2 wiki: Platform independence”, 2009. [Online]. Available: http://c2.com/cgi/wiki?PlatformIndependence

[7] R. M. Stallman et al., "Using the GNU Compiler Collection", 2016.