e-mentor
DWUMIESIĘCZNIK SZKOŁY GŁÓWNEJ HANDLOWEJ W WARSZAWIE
WSPÓŁWYDAWCA: FUNDACJA PROMOCJI I AKREDYTACJ KIERUNKÓW EKONOMICZNYCH

2014, nr 3 (55)

A. Opaliński, *Uczenie na błędach w nauczaniu programowania w systemie e-learningu*, „e-mentor” 2014, nr 3 (55), s. 38–44, DOI: 10.15219/em55.1107.

Uczenie na błędach w nauczaniu programowania w systemie e-learningu

Artur Opaliński

Jedną z kluczowych umiejętności, które muszą posiąść adepci programowania, stanowi umiejętność poprawiania kodu programu zawierającego błędy. Jest to działanie bardzo złożone, wymagające znajomości składni języka, rozumienia semantyki kodu, znajomości zasad testowania oraz rozumienia działania algorytmu. W artykule autor proponuje własną metodę kształtowania umiejętności poprawiania kodu programu wykorzystującą narzędzia do nauczania na odległość. Metoda ta jest stosowana na przedmiocie informatyka, w drugim semestrze studiów na kierunku automatyka i robotyka.

Nauka programowania komputerów w pierwszych semestrach studiów technicznych jest złożonym problemem. Mimo różnych profili kształcenia oraz stosowania różnych języków programowania studenci w badanych[1] krajach mają podobne, znaczące trudności w początkowych fazach nauki programowania. Jedną z przyczyn tego stanu rzeczy, zdiagnozowaną w artykule R. Listera i współautorów[2], jest fakt, że stworzenie poprawnie działającego programu, nawet ograniczone do samego stworzenia kodu, z pominięciem szerszych zagadnień inżynierii oprogramowania, takich jak projektowanie, testowanie i weryfikacja czy utrzymanie, jest bardzo złożonym procesem. Mimo to kształcenie i weryfikacja wiedzy adeptów programowania zazwyczaj bazują na tworzeniu całego programu rozwiązującego zadany problem.

Poniżej przedstawiono pomysł kształcenia i weryfikacji wiedzy w oparciu o poprawianie błędów w gotowych przykładowych programach, zrealizowany w grupie studentów 2. semestru kierunku *automatyka i robotyka*, uczących się programowania w języku C. Język C, choć przez środowisko akademickie uznawany jest za gorzej nadający się do celów dydaktycznych, jest wciąż szeroko nauczany ze względu na dużą praktyczną przydatność, szczególnie w pewnych niszach, do których zdecydowanie należy automatyka i robotyka.

Przedstawiony pomysł powinien być traktowany wyłącznie jako jeden z elementów kształcenia, gdyż skupia się tylko na części umiejętności niezbędnych do opanowania wybranych aspektów programowania komputerów.

Błędy w programowaniu

Program komputerowy stanowi sformalizowany opis rozumowania człowieka, który go stworzył. Jednak człowiek zwykle nie myśli dostatecznie precyzyjnie, stąd nawet doświadczeni programiści popełniają błędy w kodzie. Istnieją różne ogólne taksonomie błędów programistycznych[3]. Jedną z ogólniejszych jest następujący podział[4]:

[1] M. McCracken i in., *A multi-national, multi-institutional study of assessment of programming skills of first-year CS students*, [w:] *Proceeding of ITiCSE-WGR '01 Working group reports from ITiCSE on Innovation and technology in computer science education*, New York 2001, s. 125–180; R. Lister i in., *A multi-national study of reading and tracing skills in novice programmers*, [w:] *Proceeding of the ITiCSE-WGR '04 Working group reports from ITiCSE on Innovation and technology in computer science education*, New York 2004, s. 119–150; M.H. Nienaltowski, M. Pedroni, B. Meyer, *Compiler error messages: what can help novices?*, [w:] *SIGCSE '08 Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, New York 2008, s. 168–172; A. Robins, J. Rountree, N. Rountree, *Learning and teaching programming: A review and discussion*, „Computer Science Education" 2003, Vol. 13, No. 2, s. 137–172.

[2] R. Lister i in., dz.cyt.

[3] A. Robins, J. Rountree, N. Rountree, dz.cyt.; K. Ala-Mutka, *Problem in Learning and Teaching Programming – a literature study for developing visualizations in the Codewitz-Minerva project*, Institute of Software Systems, Tampere University of Technology, 2003, www.cs.tut.fi/~edge/literature_study.pdf, [16.06.2014]; E. Soloway, J. Spohrer, *Studying the Novice Programmer*, Lawrence Erlbaum Associates, Hillsdale 1989; B. Beizer, *Software Testing Techniques*, wyd. 2, International Thomson Computer Press, 1990; C.E. Landwehr, A.R. Bull, J.P. McDermott, W.S. Choi, *A Taxonomy of Computer Program Security Flaws*, „ACM Computing Surveys" 1994, Vol. 26, No. 3, s. 211–254; K. Tsipenyuk, B. Chess, G. McGraw, *Seven pernicious kingdoms: a taxonomy of software security errors*, „IEEE Transactions on Security & Privacy" 2005, Vol. 3, No. 6, s. 81–84; H. Krawczyk, B. Wiszniewski, *Classification of software defects in parallel programs*, Technical Report, 1994; V. Vipindeep, Pankaj Jalote, *List of Common Bugs and Programming Practices to avoid them*, 2005.

[4] *Know Your Bugs: Three Kinds of Programming Errors*, Visual Studio 2008, Visual Basic Guided Tour, http://e-mentor.pl/2254, [16.06.2014].

- Błędy składniowe (*syntax errors*) – uniemożliwiające uruchomienie programu. Najczęstszą przyczyną jest zrobienie literówki podczas pisania; często także słaba znajomość składni instrukcji przez studenta. Środowisko programistyczne zazwyczaj wskazuje miejsce wystąpienia błędu składniowego z dokładnością do pojedynczej linii kodu, jednak komunikat o błędzie – choć precyzyjny – jest lakoniczny i sformułowany technicznym językiem angielskim, trudnym do zrozumienia dla początkującego programisty.
- Błędy wykonania (*run-time errors*) – występujące w czasie działania programu, powodujące jego zatrzymanie. Zwykle wynikają z próby realizacji polecenia, które jest niemożliwe do wykonania, np. dzielenia przez zero. Jeśli są uzależnione od konkretnych wartości danych, nie występują przy każdym uruchomieniu programu. Nie mogą tym samym zostać wykryte (jak błędy składniowe) przez komputer przed uruchomieniem programu. Ich wykrycie w trakcie działania powoduje komunikaty zależne od języka programowania. W niektórych językach (Java, Python) komunikaty te mogą dość jednoznacznie wskazywać przyczynę, zaś w przypadku programowania w języku C częstym przypadkiem jest tylko bardzo oględny komunikat systemu operacyjnego.
- Błędy logiczne – powodujące, że program nie realizuje tych zadań, które programista miał na myśli. W przeciwieństwie do błędów składni, pozwalają uruchamiać program. W przeciwieństwie do błędów wykonania, nie powodują awaryjnego zatrzymania programu. Jeśli są uzależnione od konkretnych wartości danych, nie występują przy każdym uruchomieniu programu. Niekiedy błędy ujawniające się tylko przy pewnej kombinacji danych wejściowych określa się jako ujawniające się z opóźnieniem (*latent errors*).

W przypadku programowania w języku C dodatkowo można wyróżnić błędy uniemożliwiające działanie konsolidatora (linkera); choć ich przyczyną są często literówki lub pominięcia w tekście, powodujące niemożność uruchomienia programu, to wyświetlane komunikaty nie wskazują jednoznacznie błędnych linii w kodzie programu.

Błędy składniowe według powyższego podziału są na ogół najłatwiejsze do zdiagnozowania i usunięcia, zaś błędy logiczne – najtrudniejsze.

Ponieważ podczas tworzenia oprogramowania błędy w praktyce inżynierskiej są praktycznie nieuniknione, również studenci muszą opanować pracę w zwykłym programistycznym cyklu obejmującym: edycję kodu, usuwanie błędów składniowych oraz uruchamianie i testowanie programu w celu znalezienia błędów wykonania i błędów logicznych (rys.1). Tę pracę wspomagają narzędzia programistyczne, w zestawach zwanych IDE (*Integrated Development Environment*, zintegrowane środowisko programistyczne). Błędy składniowe wykrywane są przez kompilator lub linker. Błędy wykonania wykrywane są przez środowisko uruchomieniowe (*runtime*). Na wstępnych kursach programowania nie uczy się na ogół wykorzystania debuggera.

Rysunek 1. Cykl tworzenia kodu programu

Źródło: opracowanie własne.

Błędy wykrywane przez powyższe narzędzia są jednak frustrujące dla studentów[5]. Po części może być za to odpowiedzialne środowisko szkoły, w którym błądzenie nie jest na ogół rozpatrywane jako naturalny efekt uboczny dążenia do celu i rozwoju, lecz jest karane. Dodatkowymi źródłami frustracji są jednak na pewno:

- lakoniczność komunikatów o błędach składniowych,
- złożoność błędów wykonania i logicznych.

Drugi z wymienionych czynników jest o wiele bardziej złożony niż pierwszy, zaś do jego zniwelowania konieczne jest trudne do werbalnego przekazania studentom *know-how.*

Obecny stan uczenia programowania na błędach

Uczenie studentów podstaw rozpoznawania i usuwania błędów ma stosunkowo niewielką bibliografię. Dokonano kilku prób stworzenia taksonomii lub przynajmniej statystyk błędów popełnianych przez adeptów programowania[6], ujmując w nich nawet

[5] S.M. Thompson, *An Exploratory Study of Novice Programming Experiences and Errors*, praca magisterska, University of Victoria, 2004.

[6] E.M. Nalaka, S. Edirisinghe, *Teaching Students to Identify Common Programming Errors Using a Game*, [w:] *SIGITE'08 Proceedings of the 9th ACM SIGITE Conference on Information Technology Education*, 2008, s. 95–98; M. Hristova, A. Misra, M. Rutter, R. Mercuri, *Identifying and Correcting Java Programming Errors for Introductory Computer Science Students*, [w:] *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, ACM Press, Reno 2005, s. 153–156; I. Tuugalei, Ch. Mow, *Analyses of Student Programming Errors In Java Programming Courses*, „Journal of Emerging Trends in Computing and Information Sciences" 2012, Vol. 3, No. 5, s. 739 –749; M. Ahmadzadeh, D. Elliman, C. Higgins, *An analysis of patterns of debugging among novice computer science students*, [w:] *Proceeding ITiCSE '05 Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, s. 84–88, New York 2005.

błędy występujące w bardziej zaawansowanych konstrukcjach, takich jak zagnieżdżone pętle, wywołania rekurencyjne, tablice[7]. W pracy pt. *Teaching Students to Identify Common Programming Errors Using a Game*[8] opisano próbę nauczenia komunikatów o błędach składniowych typowych dla programów zapisanych w języku Java. W tym celu opracowano grę komputerową, w której krótkie błędne fragmenty kodu opadają z góry na dół ekranu, niczym klocki w grze Tetris. Zadaniem grającego studenta jest kliknięcie w ograniczonym czasie na właściwy komunikat o błędzie znajdujący się u dołu ekranu, tj. na ten komunikat, który byłby wywołany danym fragmentem kodu. Wydaje się, że autor gry próbował w ten sposób zainteresować studentów i skupić ich uwagę na fragmentach kodu. Jednak metodycznie rzecz biorąc, odtwarzanie z pamięci komunikatów o błędach nie jest kluczowym elementem programowania, a przesuwanie się kodu przed oczami nie stanowi realnego doświadczenia programistycznego. Ponadto podczas usuwania błędów kierunek myślenia jest odwrotny, tj. na podstawie komunikatu o błędzie wyszukuje się wadliwy kod, a nie odwrotnie. Korzystnym efektem zastosowania gry z perspektywy studentów było stworzenie przyjaznego środowiska do identyfikowania błędów – dużo łatwiejszego w obsłudze niż narzędzia IDE.

M. Hristova, A. Misra, M. Rutter i R. Mercuri[9] oraz Y. Morimoto, K. Kurasawa, S. Yokoyama, M. Ueno i Y. Miyadera[10] opisują opracowanie oprogramowania, które rozpoznawało w kodzie studentów typowe błędy, zebrane wcześniej przez autorów w katalogu, i udzielało programującym studentom porad na temat znaczenia i sposobów usunięcia tych błędów. Porady były obszerniejsze niż te oferowane przez IDE. Efektywność narzędzia nie była mierzona.

Badania opisane w pracy pt. *Analyses of Student Programming Errors In Java Programming Courses*[11] pozwoliły zidentyfikować najczęstsze błędy składniowe oraz niektóre błędy wykonania typowe dla studentów programujących w języku Java przez kilka kolejnych semestrów nauki. Autorka wskazanego opracowania wyraża opinię, że większość zidentyfikowanych błędów to błędy składniowe, które wynikają z niestarannego pisania kodu, a jednocześnie przyznaje, że nie uwzględniała czasu na poprawienie kodu – co wskazuje na to, że pominęła kwestię rozpoznawania i poprawiania błędów, która jest istotnym elementem nauki programowania.

Pewną próbę wykorzystania informacji diagnostycznej o błędach w nauczaniu studentów opisano w opracowaniu pt. *Compiler error messages: what can help novices?*[12]. W tym przypadku rozważono kilka języków programowania oraz kilka IDE w celu dobrania narzędzi generujących komunikaty o błędach, które przyczyniają się do skuteczniejszego usuwania błędów przez studentów. Badanie przeprowadzono na dwóch grupach studentów w dwóch różnych ośrodkach, nie uzyskując potwierdzenia, że dłuższe komunikaty są korzystniejsze dla początkujących programistów. Autorzy omawianego opracowania sugerują, że istnieje potrzeba doprecyzowania ankiet badawczych i skupienia się na zróżnicowanej charakterystyce (zawartości) komunikatów o błędach, a nie na ich długości. Wydają się tym samym pomijać inne niż tylko rozumienie komunikatów aspekty wyszukiwania błędów w kodzie.

Z kolei system działający w czasie rzeczywistym, opisany w opracowaniu pt. *Analysis of Errors – A Support System for Teachers to Analyze the Error Occurring to a Novice Programmer*[13], automatycznie analizuje komunikaty o błędach w kodzie studentów i udziela programującym poszerzonych informacji na temat możliwego rozwiązania problemów, zaś nauczycielowi oferuje podsumowanie błędów popełnianych przez studentów. Artykuł nie zawiera oceny efektywności rozwiązania; można się spodziewać, że ułatwia ono nauczycielowi wspomaganie dużej grupy uczniów, jednak samo dostarczenie poszerzonych komunikatów może nie stanowić skutecznej pomocy, podobnie jak to opisali M.H. Nienaltowski, M. Pedroni i B. Meyer [14].

C-Teacher – nowa metoda uczenia programowania na błędach

Wyniki badań[15] sugerują, że efektywną metodą uczenia programowania na podstawie błędów może być prezentowanie niewielkich, kompletnych, lecz zawierających błędy programów, które studenci muszą poprawić. Taki system nauki programowania autor nazwał C-Teacher, ze względu na zastosowanie do nauki języka C. Program do poprawienia zawiera kontrolowaną liczbę (do 5, przyjęte arbitralnie w zależności od poziomu trudności) błędów:

[7] D. Kopec, G. Yarmish, *Revisiting Novice Programmer Errors*, „ACM SIGCSE Bulletin" 2007, Vol. 39, No. 2, s. 131–137.
[8] E.M. Nalaka, S. Edirisinghe, dz.cyt.
[9] M. Hristova, A. Misra, M. Rutter, R. Mercuri, dz.cyt.
[10] Y. Morimoto, K. Kurasawa, S. Yokoyama, M. Ueno, Y. Miyadera, *A Support System for Teaching Computer Programming Based on the Analysis of Compilation Errors*, Sixth IEEE International Conference on Advanced Learning Technologies (ICALT'06), materiały konferencyjne, Kerkrade, 5–7.07.2006, s. 103–105.
[11] I. Tuugalei, Ch. Mow, *Analyses of Student Programming Errors In Java Programming Courses*, „Journal of Emerging Trends in Computing and Information Sciences" 2012, Vol. 3, No. 5, s. 739–749.
[12] M.H. Nienaltowski, M. Pedroni, B. Meyer, dz.cyt.
[13] A. Bhawkar, R. Belsare, F. Gandhi, P. Somani, *Analysis of Errors – A Support System for Teachers to Analyze the Error Occurring to a Novice Programmer*, „International Journal of Computer Science and Network" 2013, Vol. 2, No. 5, s. 37–40.
[14] M.H. Nienaltowski, M. Pedroni, B. Meyer, dz.cyt.
[15] M. McCracken i in., dz.cyt.; R. Lister i in., dz.cyt.; M. Ahmadzadeh, D. Elliman, C. Higgins, dz.cyt.

- składniowych, obejmujących brak średników kończących instrukcje, brak nawiasów do pary, nadmiarowe nawiasy niepasujące do pary, nazwy standardowych funkcji języka C zawierające literówki, brakujące deklaracje zmiennych, niewłaściwy typ zmiennych w wyrażeniach;
- wykonania, obejmujących błędne operatory, błędny priorytet działań, niewłaściwe typy danych liczbowych powodujące zaokrąglenia, nadmiarowe średniki powodujące nieefektywność instrukcji warunkowych i pętli, użycie niezainicjalizowanych zmiennych;
- logicznych, obejmujących niewłaściwą kolejność dwóch sąsiadujących instrukcji, niewłaściwe wyrażenia, użycie dwóch zmiennych w zamienionych rolach.

Każdy nietrywialny program wymagający poprawy zawiera w komentarzu opis pożądanego działania oraz propozycję przypadków testowych, tj. zestawu danych, dla którego powinien być przetestowany. Przypadki testowe są dobrane tak, aby powodowały uwidocznienie błędów i tym samym zapobiegały ich opóźnionemu ujawnianiu czy zupełnemu niedostrzeżeniu. Rozwiązując zadanie, student ma do dyspozycji standardowe IDE, z którym na co dzień pracuje na zajęciach i w domu.

Takie postawienie problemu powoduje, że student ma do czynienia z sytuacją bardzo realną, a jednocześnie kontrolowaną. Realny jest zestaw narzędzi i ich komunikaty o błędach, realne jest także poprawianie istniejącego już programu o oczywistych założeniach; wymaga to czytania i rozumienia kodu oraz umiejętności jego uruchamiania i testowania. Jednocześnie sytuacja jest uproszczona w stosunku do przypadku pisania własnego programu od podstaw: student nie wymyśla rozwiązania problemu, stopień komplikacji kodu jest ograniczony, ograniczone są także liczba i rodzaj błędów. Jednocześnie wyklucza się przypadkowe napisanie bardzo prostego programu od razu bez błędu, co mogłoby przynieść studentowi fałszywe poczucie, że umie programować.

IDE spełnia tu do pewnego stopnia korygującą rolę nauczyciela, gdyż – choć stosuje wspominane lakoniczne komunikaty – pozwala nawet słabemu studentowi samodzielnie przynajmniej weryfikować wyniki własnej pracy w zakresie poprawności bądź jej braku, zanim zostaną one przekazane do sprawdzenia nauczycielowi. Dzięki komunikatom IDE student może także w trakcie zajęć zadawać nauczycielowi precyzyjniejsze pytania.

Możliwości weryfikacji w systemie nauczania C-Teacher

Weryfikacja w systemie nauczania C-Teacher polega na sprawdzeniu, czy program poprawiony przez studenta nie zawiera błędów składniowych (czy możliwa jest jego kompilacja i konsolidacja w IDE) oraz błędów wykonania i logicznych (czy działa poprawnie dla opisanych i nieopisanych przypadków testowych).

Testowanie na nieopisanych, czyli nieznanych studentowi przypadkach testowych jest konieczne, aby zapobiec przedstawianiu rozwiązań, które działają poprawnie wyłącznie dla przypadków testowych. Ponieważ poprawiany prosty program służy np. do dodawania liczb lub do znalezienia najmniejszej z liczb, potencjalne przypadki testowe są liczne i poprawny sposób działania jest oczywisty dla studenta.

Weryfikacja dużej liczby programów zawierających czasami bardzo drobne literówki lub subtelne błędy logiczne byłaby trudna dla nauczyciela, nawet przy wykorzystaniu IDE, ze względu na znaczną liczbę przypadków testowych i konieczność precyzyjnego porównywania wyników uzyskanych z oczekiwanymi.

Do automatyzacji tego typu zadań służą webowe systemy typu *Online Judge* (OJ, sędzia internetowy)[16]. Systemy te umożliwiają studentom zakładanie indywidualnych kont i przekazywanie rozwiązań zadań, które są następnie automatycznie kompilowane, uruchamiane dla zdefiniowanych przypadków testowych i oceniane. Systemy typu OJ przeznaczone są jednak raczej do przeprowadzania konkursów niż do nauki i dlatego nie odpowiadają wszystkim wymaganiom nauczania, w szczególności zakładane na nich konta nie są powiązane z uczelnianymi kontami studentów w elektronicznych systemach zarządzania kształceniem i nie pozwalają na weryfikację kont. Same systemy OJ nie oferują równie bogatych możliwości prezentowania treści czy weryfikacji wiedzy jak systemy LMS.

W Harbińskim Instytucie Techniki powstał zintegrowany moduł OJ[17] dla platformy Moodle. Niestety od 2012 r. moduł ten nie jest już rozwijany[18] i nie współpracuje z aktualnymi wersjami Moodle, poczynając od wersji 2.3[19]. Nie jest także świadczona pomoc instytucjom, które chciałyby go wykorzystywać[20]. Wszystkie te czynniki podważają możliwości wykorzystania tego modułu do oceniania studentów.

[16] Systemy typu *Online Judge*: UVa Online Judge, http://uva.onlinejudge.org; ACM International Collegiate Programming Contest, http://icpc.baylor.edu; International Olympiad in Informatics, http://www.ioi2013.org, SPOJ, Sphere Online Judge, URL: http://pl.spoj.com, [16.06.2014].

[17] S. Zhigang, S. Xiaohong, Z. Ning, Ch. Yanyu, *Moodle Plugins for Highly Efficient Programming Courses*, 1st Moodle Research Conference, materiały konferencyjne, Heraklion, 14–15.09.2012, s. 157–163.

[18] Strona projektu *Moodle Online Judge* ze statystykami prowadzonych prac, http://www.ohloh.net/p/moodle-online-judge, [16.06.2014].

[19] S. Zhigang, S. Xiaohong, Z. Ning, Ch. Yanyu, dz.cyt.

[20] A. Tashakor, post o problemach w *Moodle Online Judge*, https://moodle.org/mod/forum/discuss.php?d=193880; Karthikeya Acharya, post o problemach w *Moodle Online Judge*, https://moodle.org/mod/forum/discuss.php?d=206571.

Weryfikacja z oprogramowaniem ECOLS

Wiele przydatnych cech pozwalających na weryfikację umiejętności poprawiania błędów programistycznych przez studentów posiada moduł Quizz w Moodle, który na dodatek można wykorzystywać w ramach zajęć komplementarnych na uczelni, w celu kontroli samodzielności pracy studenta. Moduł ten oferuje:

- zbiór pytań z możliwością losowania odmiennych zestawów dla poszczególnych uczestników quizu,
- limitowany czas realizacji zadania,
- dostęp do zadania ograniczony czasowo i za pomocą hasła oraz adresu IP.

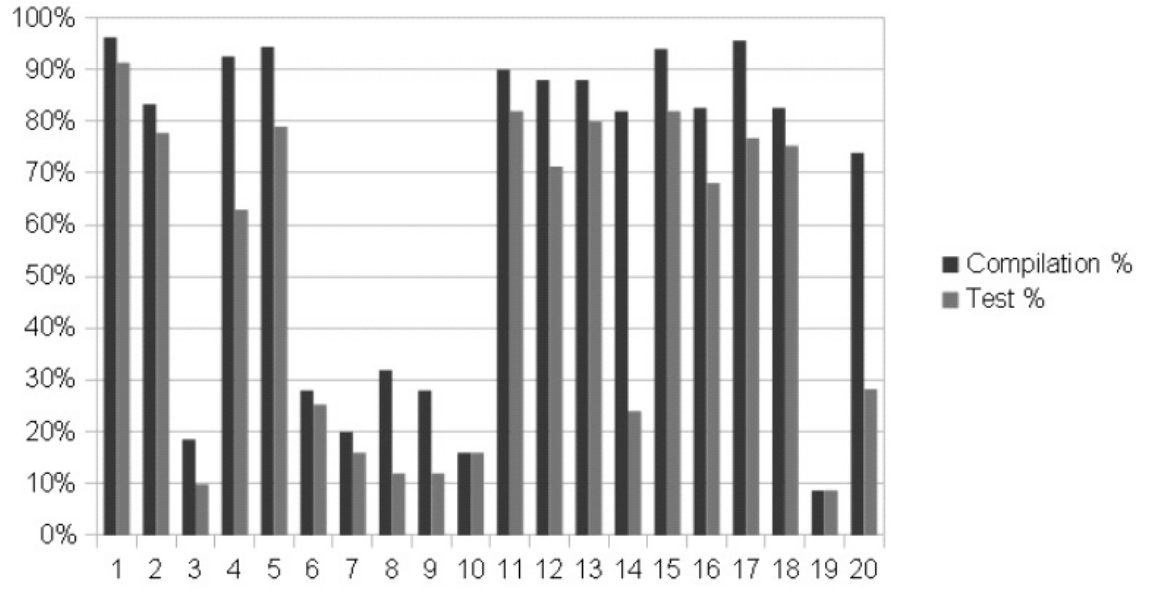
Nadzór nauczyciela, zapewniający kontrolę samodzielności pracy studenta, jest tu dość istotny, gdyż po poprawieniu programy składane przez studentów nie wykazują wielkich różnic, co nie pozwala na efektywnie ich porównywanie w ramach automatycznej kontroli antyplagiatowej.

Stosując pytania typu „esej", można w treści pytania podawać kod zawierający błędy i opis wymagań czy przypadków testowych. Kod ten użytkownik może skopiować i wkleić do swojego IDE, zaś poprawioną wersję programu wkleić jako odpowiedź na to pytanie.

Niestety, pytania typu „esej" nie są w Moodle oceniane automatycznie, a w szczególności nie można się spodziewać, że oceniona zostanie poprawność kodu zawartego w odpowiedzi. Autor opracował w związku z tym własne oprogramowanie ECOLS[21], które zapewnia automatyczne ocenianie kodu. Oprogramowanie to kolejno:

- pobiera z serwera Moodle zawartość bazy użytkowników zapisanych na kurs;
- pobiera z serwera Moodle zawartość pytań oraz odpowiedzi z podanego quizu i dzieli je na odrębne pliki;
- przyporządkowuje powstałe pliki do użytkowników z bazy;
- wylicza skrót (*hash*) z pliku zawierającego pytanie;
- posługując się powyższym skrótem, odnajduje w bazie C-Teacher opisy przypadków testowych oraz poprawny kod wzorcowy;
- poprzez kompilację i konsolidację sprawdza, czy nie ma błędów składniowych w kodzie złożonym przez studenta;
- uruchamia skompilowany program złożony przez studenta, kontrolując czas jego wykonania; zbyt długi czas wykonania interpretowany jest jako zawieszenie się programu i dyskwalifikuje rozwiązanie;
- uruchamia program wzorcowy, podając na jego standardowe wejście dane z tych samych przypadków testowych, które są wprowadzane do programu złożonego przez studenta, i porównuje wyniki na ich standardowych wyjściach;
- sumuje wyniki uzyskane za poprawne wykonanie poszczególnych przypadków testowych, tworząc ostateczną ocenę pracy studenta.

Wykres 1. Wyniki uzyskane przez studentów. Na osi poziomej – numery zadań. Na osi pionowej – odsetek programów poprawnych składniowo (Compilation%) oraz poprawnych logicznie (Test%)



Źródło: opracowanie własne.

[21] A. Opalinski, *Integrating web site services into application through user interface*, III Konferencja TEWI, materiały konferencyjne, Politechnika Łódzka, Łódź, 03.07.2012.

Po połowie pierwszego semestru nauczania programowania w języku C 160 studentów w ciągu jednego tygodnia pisało test, w czterech różnych terminach, pod nadzorem nauczyciela. Grupa studentów w każdym terminie dostawała odmienny zestaw pięciu pytań. Każdy indywidualnie rozwiązywał zadania z zestawu przypisanego do jego grupy. Kolejność prezentowania zadań z zestawu była losowana oddzielnie dla każdego uczestnika.

Cała przygotowana pula liczyła zatem 20 zadań, w postaci odmiennych fragmentów kodu zawierającego błędy. Dla każdego zadania odnotowano odsetek poprawnych kompilacji i konsolidacji, tj. rozwiązań, w których nie występowały błędy składniowe, oraz odsetek poprawnie działających przypadków testowych. Rezultaty przedstawia wykres 1. Wynika z niego, że odsetek programów bez błędów składniowych zawsze jest nie niższy niż odsetek poprawnie rozwiązanych przypadków testowych. Jest to poprawne, gdyż brak błędów składniowych jest warunkiem koniecznym uruchomienia programu.

Studenci rozwiązujący zadania 6–10 uzyskali słabsze wyniki, co pokrywa się z innymi składnikami ich oceny końcowej z przedmiotu (podczas realizacji przedmiotu oceniane były także: rozwiązywanie wejściówek, rozwiązywanie testów podczas sprawdzianów zaliczających, aktywność na zajęciach, liczba i jakość prac domowych, umiejętność zakodowania algorytmu schematu blokowego w języku programowania podczas testu). Nie prowadzono jednak analizy tego, czy przypisany do ich terminu zestaw pytań charakteryzował się odmiennym poziomem trudności od pozostałych zestawów. Losowa próbka 10 proc. studentów została dodatkowo oceniona przez nauczyciela z użyciem IDE, w celu weryfikacji programu ECOLS. Nie zaobserwowano różnic między ocenianiem „ręcznym" z użyciem IDE a ocenianiem automatycznym.

Podsumowanie

W artykule autor wyróżnił pewną grupę umiejętności w ramach nauki programowania i zdefiniował ją jako nauczany element, poprzez określenie:

- założeń wstępnych dotyczących znajomości składni i rozumienia pojęcia programowania,
- zadania służącego do rozwijania wybranej umiejętności, którym było poprawianie istniejących przykładów kodu zawierającego błędy, w typowym środowisku IDE,
- zadania weryfikującego i sposobu jego realizacji w środowisku nauczania na odległość, zapewniającym kontrolę samodzielnej pracy studenta,
- metody i narzędzia do oceny zadania weryfikującego, zapewniających jednolitość sprawdzania i sprawdzanie prac w masowej skali.

Autor wykazał w praktyce dydaktycznej, że pomysł nauczania wybranych umiejętności programistycznych poprzez poprawianie błędów w istniejącym kodzie jest wykonalny, opracowując konfigurację quizu Moodle, zestaw pytań oraz narzędzie do automatycznego oceniania.

Przyszłe badania mogą dotyczyć analizy błędów najczęściej popełnianych przez studentów w systemie C-Teacher, być może z podziałem na nieoceniany okres treningowy oraz na pisanie testu na ocenę. Ponadto wartościowe będzie poznanie opinii studentów na temat systemu C-Teacher.

Bibliografia

M. Ahmadzadeh, D. Elliman, C. Higgins, *An analysis of patterns of debugging among novice computer science students*, [w:] *Proceeding ITiCSE '05 Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, s. 84–88, New York 2005.

K. Ala-Mutka, *Problem in Learning and Teaching Programming – a literature study for developing visualizations in the Codewitz-Minerva project*, Institute of Software Systems, Tampere University of Technology, 2003, www.cs.tut.fi/~edge/literature_study.pdf.

B. Beizer, *Software Testing Techniques*, wyd. 2, International Thomson Computer Press, 1990.

B. Bereza-Jarociński, B. Szomański, *Inżynieria oprogramowania. Jak zapewnić jakość tworzonym aplikacjom*, Helion, Gliwice 2009.

A. Bhawkar, R. Belsare, F. Gandhi, P. Somani, *Analysis of Errors – A Support System for Teachers to Analyze the Error Occurring to a Novice Programmer*, „International Journal of Computer Science and Network" 2013, Vol. 2, No. 5, s. 37–40.

M. Hristova, A. Misra, M. Rutter, R. Mercuri, *Identifying and Correcting Java Programming Errors for Introductory Computer Science Students*, [w:] *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, ACM Press, Reno 2005, s. 153–156.

Know Your Bugs: Three Kinds of Programming Errors, Visual Studio 2008, Visual Basic Guided Tour, http://e-mentor.pl/2254.

D. Kopec, G. Yarmish, *Revisiting Novice Programmer Errors*, „ACM SIGCSE Bulletin" 2007, Vol. 39, No. 2, s. 131–137.

H. Krawczyk, B. Wiszniewski, *Classification of software defects in parallel programs*, Technical Report, 1994.

C.E. Landwehr, A.R. Bull, J.P. McDermott, W.S. Choi, *A Taxonomy of Computer Program Security Flaws*, „ACM Computing Surveys" 1994, Vol. 26, No. 3, s. 211–254.

R. Lister i in., *A multi-national study of reading and tracing skills in novice programmers*, [w:] *Proceeding of the ITiCSE-WGR '04 Working group reports from ITiCSE on Innovation and technology in computer science education*, New York 2004, s. 119–150.

M. McCracken i in., *A multi-national, multi-institutional study of assessment of programming skills of first-year CS students*, [w:] *Proceeding of ITiCSE-WGR '01 Working group reports from ITiCSE on Innovation and technology in computer science education*, New York 2001, s. 125–180.

Y. Morimoto, K. Kurasawa, S. Yokoyama, M. Ueno, Y. Miyadera, *A Support System for Teaching Computer Programming Based on the Analysis of Compilation Errors*, Sixth IEEE International Conference on Advanced Learning Technologies (ICALT'06), materiały konferencyjne, Kerkrade, 5–7.07.2006, s. 103–105.

E.M. Nalaka, S. Edirisinghe, *Teaching Students to Identify Common Programming Errors Using a Game*, [w:] *SIGITE '08 Proceedings of the 9th ACM SIGITE Conference on Information Technology Education*, 2008, s. 95–98.

M.H. Nienaltowski, M. Pedroni, B. Meyer, *Compiler error messages: what can help novices?*, [w:] *SIGCSE '08 Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, New York 2008, s. 168–172.

A. Opalinski, *Integrating web site services into application through user interface*, III Konferencja TEWI, materiały konferencyjne, Politechnika Łódzka, Łódź, 03.07.2012.

A. Robins, J. Rountree, N. Rountree, *Learning and teaching programming: A review and discussion*, „Computer Science Education" 2003, Vol. 13, No. 2, s. 137–172.

E. Soloway, J. Spohrer, *Studying the Novice Programmer*, Lawrence Erlbaum Associates, Hillsdale 1989.

S.M. Thompson, *An Exploratory Study of Novice Programming Experiences and Errors*, praca magisterska, University of Victoria, 2004.

K. Tsipenyuk, B. Chess, G. McGraw, *Seven pernicious kingdoms: a taxonomy of software security errors*, „IEEE Transactions on Security & Privacy" 2005, Vol. 3, No. 6, s. 81–84.

I. Tuugalei, Ch. Mow, *Analyses of Student Programming Errors In Java Programming Courses*, „Journal of Emerging Trends in Computing and Information Sciences" 2012, Vol. 3, No. 5, s. 739–749.

V. Vipindeep, Pankaj Jalote, *List of Common Bugs and Programming Practices to avoid them*, 2005.

S. Zhigang, S. Xiaohong, Z. Ning, Ch. Yanyu, *Moodle Plugins for Highly Efficient Programming Courses*, 1st Moodle Research Conference, materiały konferencyjne, Heraklion, 14–15.09.2012, s. 157–163.

Learning by mistakes in teaching programming through distance learning

Despite employing various programming languages on different course majors, teaching novice programmers on technical departments is a complex issue in many countries worldwide. References have been given to the analyses of the most common programming errors encountered by students, and to the attempts to aid students in eradicating these errors. Aiding students was based each time on automatic teacher's advices to common errors, so it encompassed easing teacher's efforts as well.

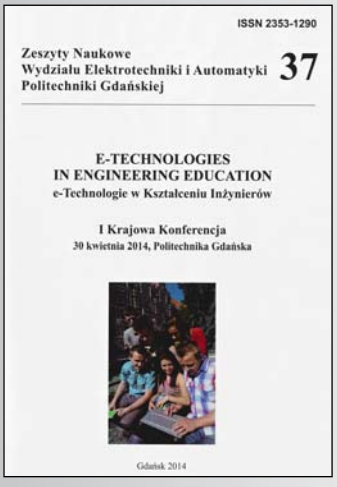
One of the key skill that novice programmers have to master is to independently correct errors in code. While this is only a part of the required skills, it requires the knowledge of language syntax and testing rules as well as the understanding of code semantics and algorithm.

The author proposes in the paper a new method of teaching the skills of correcting code, which is based on correcting ready-made code samples which contain purposely-made errors. The supporting tool is the typical IDE that students use in their everyday work. In this way a controlled learning environment is created, where the number and the type of errors, as well as the error messages generated by IDE are predefined. Such an environment is much better defined that during a typical assignment where the student is supposed to write a program from scratch.

The author suggests using distance learning tools for teaching and for skills verification with the above described method. The new method has been applied to the Computer Science course taught in the Process Control and Robotics major.

Autor jest doktorem inżynierem w dziedzinie elektrotechniki, adiunktem na Politechnice Gdańskiej. Narzędzia nauczania na odległość stosuje od 4 lat we wszystkich prowadzonych przez siebie kursach z zakresu programowania. Jego zainteresowania naukowe dotyczą agentów programowych oraz sztucznej inteligencji.

POLECAMY



E-technologies in Engineering Education
„Zeszyty Naukowe Wydziału Elektrotechniki i Automatyki Politechniki Gdańskiej" 2014, nr 37

Zapraszamy do zapoznania się z „Zeszytami Naukowymi Wydziału Elektrotechniki i Automatyki Politechniki Gdańskiej" zawierającymi zestaw referatów z I Krajowej Konferencji pt. *e-Technologie w Kształceniu Inżynierów*, która odbyła się 30 kwietnia 2014 roku. Poruszane zagadnienia to m.in. kursy online na platformie edX, kwestia ochrony danych osobowych w e-learningu, nowe rozwiązania informatyczne, wykorzystanie programów na licencji GNU, podejście studentów do e-oceniania.
Publikacja jest dostępna w wersji elektronicznej na stronie:
http://www.ely.pg.gda.pl/zn/?artykul=500.