

Henryk BUDZISZ, Krzysztof KADOWSKI, Walery SUSŁOW

Katedra Inżynierii Komputerowej, Politechnika Koszalińska

E-mail: hb@ie.tu.koszalin.pl, kadowski@hot.pl, swalover@ie.tu.koszalin.pl

Koncepcja zautomatyzowanej weryfikacji wiedzy i umiejętności z obszaru programowania komputerów

1 Wstęp

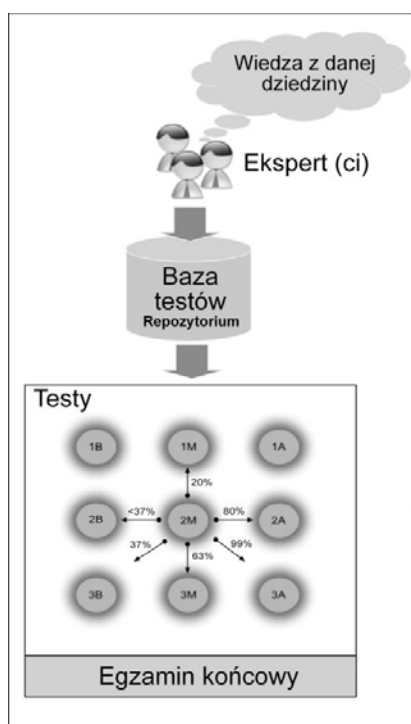
Weryfikacja wiedzy studentów może być przeprowadzona na wiele tradycyjnych sposobów. Mimo swoich niepodważalnych zalet mają one jednak wspólne wady: czasochłonność przeprowadzenia kontroli (czy sprawdzenia wyników kontroli) oraz trudności z obiektywizacją oceny. Dla obu stron procesu dydaktycznego jest to sytuacja niekorzystna. Zautomatyzowana (skomputeryzowana) kontrola staje się obecnie bardzo powszechna, przestaje być gatunkiem eksperymentu dydaktycznego [1, 2]. Zalety tego typu środków kontroli są oczywiste. W stosunku do tradycyjnych metod ilość czasu potrzebna na realizację jest niewielka, a uzyskuje się natychmiastowe oceny postępów studentów w opanowaniu przekazywanej wiedzy.

Komputeryzacja egzaminów może być rozpatrywana również jako automatyzacja pomiaru dydaktycznego, co w znacznym stopniu wspomaga nauczycieli w analizie i interpretacji wyników kształcenia. Współczesny poziom technologii informatycznych oraz istniejąca wiedza z teorii testów (testologii) dają podstawy do symulacji podmiotowego zachowania przez system testujący, możliwości takie stwarza wykorzystanie testowania adaptacyjnego i technik multimedialnych. Wszystko to przemawia za stosowaniem kontroli środkami technicznymi, gdyż można osiągnąć wyższą sprawność i efektywność procesu dydaktycznego.

2 Sformułowanie koncepcji

Programowanie komputerów i inne dyscypliny informatyczne wymagają opanowania nie tylko określonej wiedzy, ale i konkretnych umiejętności, a sprawdzanie teoretycznych i praktycznych osiągnięć jednostki, jak wiadomo, wymaga różnych metod testowania. Autorska koncepcja weryfikacji wiedzy i umiejętności programistycznych uwzględnia specyfikę przedmiotów dydaktycznych z grupy *computer science* i polega na egzaminowaniu dwuetapowym. Na etapie pierwszym sugerowany jest test wiedzy typu zamkniętego. Test ma służyć do sprawdzenia utrwalonych wiadomości z przedmiotu na poziomie *Basic/Medium*. Na drugim etapie sugerowany jest zautomatyzowany test umiejętności typu otwartego, umożliwiający sprawdzanie umiejętności studenta/ucznia w praktycznym zastosowaniu wiedzy nabytej podczas pisania aplikacji komputerowych lub ich fragmentów. Test ten odpowiada stwierdzeniu faktu nabycia kwalifikacji *Advanced/Expert*. W praktyce ta dwuetapowość ma być realizowana poprzez kroki iteracyjne, synchronizowane zakończeniem studiowania rozdziałów tematycznych danej dyscypliny.

Techniczna realizacja aplikacji testującej ma opierać się na przedstawieniu studentowi możliwości „otwartej” odpowiedzi na pytanie. Odpowiedź ma być sformułowana w jednym z języków programowania komputerów oraz natychmiast poddana weryfikacji poprzez kompilowanie i uruchomienie napisanego kodu (przy kompilacji z sukcesem). Jeśli wartości zwracane przez aplikację są identyczne z oczekiwanym szablonem, to można uznać odpowiedź za poprawną. Można pozwolić studentowi na odczytanie komunikatów kompilatora i na poprawienie odpowiedzi, jednak należy śledzić działania studenta podczas debuggowania. Ilość dokonanych poprawek, rodzaj popełnionych błędów i czas debuggowania mogą być podstawą do obliczania oceny końcowej.



Za alternatywną metodę można uznać celowe wstawianie błędnych instrukcji w kodzie programu, a zadaniem studenta przy takim testowaniu umiejętności będzie odnalezienie błędów. Zadanie można oprzeć o przewidzenie błędu, który wygeneruje kompilator oraz o analizę spójności logicznej aplikacji. Inaczej mówiąc, wstawiane do zadania testowego błędy mogą mieć charakter syntaktyczny (niezgodność z zasadami języka programowania, kompilator je wychwytuje) lub semantycznymi (błędy logiczne, brak reakcji kompilatora, można zauważyć je tylko przy wykonaniu obliczeń przykładowych).

Rys. 1. Model systemu testującego

Fig. 1. Pattern of testing system

Zadanie otwarte nie necessarily musi być sformułowane tak, by oczekiwać jako odpowiedź pełny tekst aplikacji (stosowne dla poziomu *Expert*). Są do zaakceptowania również zadania na „dopisanie” brakującego fragmentu kodu (poziom *Advanced*), którym może być funkcja lub klasa. Takie testowanie

można traktować jako weryfikację umiejętności pracy w zespole programistycznym.

Niezbędnym składnikiem aplikacji testującej musi być moduł statystyk, którego zadaniem byłoby zbieranie danych dotyczących ilości kompilacji wykonanych podczas rozwiązania testu, okresów czasu pracy nad poszczególnym zadaniem lub ilości znaków zmienianych (dopisywanych) w listingu. Na podstawie tych statystyk można dokonać korekty oceny końcowej umiejętności. Np. na podstawie rankingowania rozwiązań można uznać za lepsze te, które wykonane są przy minimalnej ilości kompilacji, przy najkrótszym dopisanym kodzie czy przy minimalnym czasie odpowiedzi.

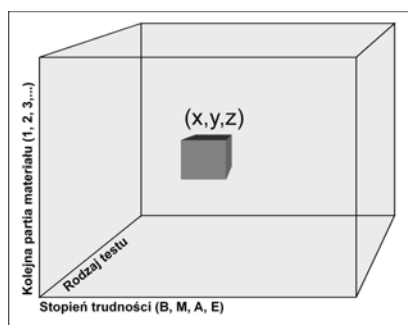
Bardzo trudnym do realizacji może być pomysł oceniania również „stylu” programowania. Należałoby opracować parser, który uwzględniałby takie elementy stylu jako formatowanie kodu, umieszczanie stosownych komentarzy czy konsekwentność przy budowie struktury aplikacji lub poprawne nazewnictwo. Odpowiedź na pytanie „Czy student zna i przestrzega dobrych zwyczajów w

programowaniu” może być bardzo korzystna do zróżnicowania ocen za kody tak samo skuteczne, ale napisane poprawnym lub niepoprawnym stylem.

System weryfikacji wiedzy i umiejętności w podejściu modelowym przedstawiono na rysunku 1 ideowo, oraz na rysunku 2 za pomocą trójwymiarowej przestrzeni testującej, po której student może poruszać się, korzystając z systemu nawigacji. System ten powinien udostępniać w zbiorze zadań testowych różne ścieżki do zaliczenia przedmiotu - student może:

- przejść pełną drogę, przechodząc wszystkie poziomy i testy na danym poziomie;
- przejść drogę tylko na wybranym poziomie szkolenia;
- wykonać zadania, sugerowane systemem według algorytmu adaptacyjnego.

W ścieżce trzeciej naukę można zakończyć finalnym testem egzaminacyjnym lub zakończyć oceną z uwzględnieniem punktacji uzyskanej na wszystkich poziomach. Tu też można zbudować algorytm oparty na przejściach ścieżkami między poszczególnymi poziomami. Takie przedstawienie umożliwia algorytmizację rozwiązań konceptualnych.



Rys. 2. Trójwymiarowy model przestrzeni testującej

Fig. 2. Three-dimensional testing pattern

Legenda do rysunku 1:

- x – stopień trudności testu (wartość 1 dla poziomu *Basic* (B), wartość 2 dla *Medium* (M), wartość 3 dla *Advanced* (A), wartość 4 dla *Expert* (E),
- y – rodzaj testu na danym poziomie (otwarty, zamknięty, adaptacyjny),
- z – kolejna partia materiału.

3 Prognozowanie osiągnięć studenta

Postaramy się dać odpowiedź na pytanie: „Jaka część umiejętności możliwa jest do opanowania na danym poziomie zaawansowania studiów, gdy umiejętności z poprzednich poziomów nie są całkiem utrwalone?” Niech u_i oznacza część umiejętności opanowanych przez studenta na poziomie i dzięki wykonaniu ćwiczeń oraz pisaniu własnych aplikacji, a w_i oznacza część wiadomości (wiedzy) z poziomu i opanowanych przez studenta. Jeśli te parametry zmieniają wartości w przedziale $[0, 1]$, to można je traktować jako prawdopodobieństwo tego, że dana umiejętność jest opanowana lub dana wiedza jest utrwalona. Wartości tych parametrów można zmierzyć przy użyciu przedstawionej metodologii testowania.

Wartość w_i zależy w pierwszej kolejności od zdolności mnemoniczych, a szczególnie od pamięci długotrwałej jednostki [3] i dlatego można przyjąć w pierwszym przybliżeniu, że nie zależy ona od i , czyli $w_i = w_0$. Wtedy $(1 - w_0)$ oznacza część wiadomości z każdego poziomu nie utrwalonych w pamięci, a $(1 - u_i)$ oznacza część umiejętności nie opanowanych na danym poziomie i .

Zakłada się, że dana umiejętność na poziomie $(i+1)$ jest niemożliwa do opanowania, jeśli student nie utrwalił w pamięci odpowiedniej (koniecznej do ukształtowania danej umiejętności) wiedzy na etapie $(i+1)$, a przy tym brak u niego koniecznych umiejętności z

poprzednich etapów kształcenia z rozwinięcia których może powstać nowa umiejętność. Logika połączenia tych dwóch zdarzeń może być opisana poprzez współczynnik $(1-w_0)(1-u_i)(1-u_{i+1})\dots(1-u_l)$. Jeśli przyjąć, że na dany etap szkolenia najbardziej wpływają umiejętności z poprzedniego etapu, to można zapisać prototyp wzoru iteracyjnego na obliczenie części umiejętności możliwej do opanowania na poziomie $(i+1)$ w postaci:

$$u_{i+1} = 1 - (1-w_0)(1-u_i) \quad (1)$$

Ze wzoru (1) bezpośrednio wynika, że brak potrzebnej wiedzy może istotnie ograniczyć osiągalny przedział umiejętności, bo przy całkowitym braku wiedzy, tj. $w_0=0$ poziom umiejętności pozostaje bez zmian, czyli $u_{i+1} = u_i$. Im większa jest wiedza studenta na danym etapie tym mniejszy wysiłek jest potrzebny do opanowania nowych umiejętności.

Kryterium opanowania umiejętności na poziomie zawodowym może przewidywać konieczność kompleksowego zastosowania jednocześnie kilku podstawowych umiejętności. Prawdopodobieństwo tego że podczas szkolenia opanowanych zostanie k umiejętności równocześnie, można zapisać jako iloczyn $u_{i1}u_{i2}u_{i3}\dots u_{ik}$. Jeżeli dla uproszczenia przyjąć, że prawdopodobieństwo opanowania każdej z k umiejętności jest jednakowe, czyli: $u_{i1} = u_{i2} = u_{i3} = u_{ik} = u_i$, to wartość prawdopodobieństwa równoczesnego opanowania k umiejętności jest u_i^k . Wówczas $(1 - u_i^k)$ oznacza prawdopodobieństwo tego, że przynajmniej jedna z potrzebnych umiejętności nie zostanie przyswojona. Przy takim założeniu wzór (1) przekształca się do postaci:

$$u_{i+1} = 1 - (1-w_0)(1-u_i^k) \quad (2)$$

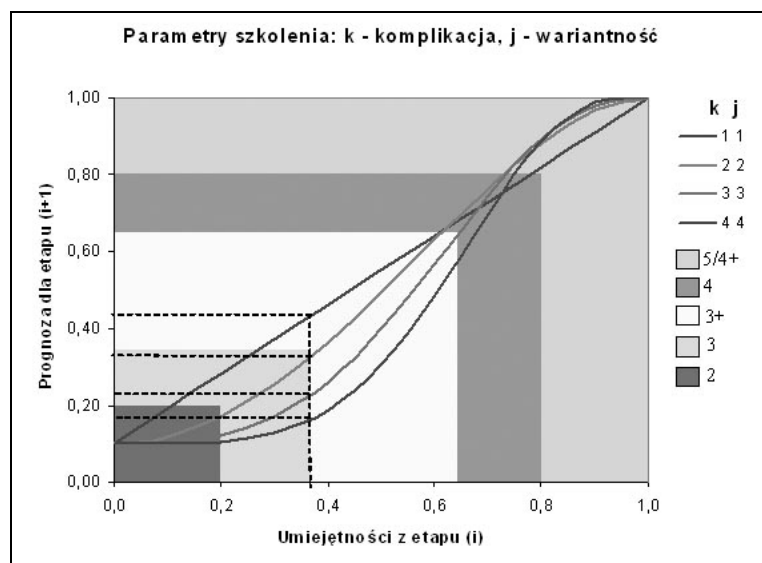
Parametr k wprowadza do wzoru iteracyjnego nową jakość - wyraźną nieliniowość prognozy osiągnięć u_{i+1} na podstawie wartości u_i . Określenie k jako *komplikacji* kursu szkolenia pozwoli stwierdzić, że istnieje minimalny konieczny poziom umiejętności u_{imin} (patrz rysunek 3), przy którym szkolenie na poziomie $(i+1)$ w ogóle ma sens.

Poza tym, z praktyki programowania komputerów wiadomo, że istnieją możliwości skutecznego napisania programu na j sposobów, na przykład wykonanie niektórych obliczeń można zaprogramować prawie tak samo dobrze poprzez wykorzystanie algorytmu iteracyjnego lub poprzez zastosowanie rekurencji ($j=2$). Może to oznaczać wystarczalność opanowania nawet tylko jednej z j umiejętności do skutecznego wykonania aplikacji, a co za tym idzie i pracy zaliczeniowej. W związku z tymi okolicznościami należałoby wprowadzić do wzoru (2) prawdopodobieństwo tego, że przynajmniej jedna z wzajemnie równoznacznych umiejętności kompleksowych jest opanowana. Prawdopodobieństw tego, że żadna z potrzebnych umiejętności nie została przyswojona zapiszemy jako $(1-u_i^k)^j$, co skutkuje wzorem:

$$u_{i+1} = 1 - (1-w_0)(1-u_i^k)^j \quad (3)$$

Parametr modelu j można traktować jako wskaźnik *wariantowości* materiału szkoleniowego. Im bogatszy materiał szkoleniowy, tym większe prawdopodobieństwo opanowania przez dobrego (pracowitego) studenta przynajmniej jednego ze sposobów wykonania projektów (patrz rysunek 3).

Liniowa prognoza maksymalnych osiągnięć na etapie $(i+1)$ odnosi się wyłącznie do szkoleń prymitywnych ($k=j=1$), które nie przewidują kształtowania kompleksowych umiejętności oraz nie dają wariantów na wykonanie zadań. Dla bardziej skomplikowanych kursów istnieje minimalny poziom umiejętności na wejściu, przy którym można spodziewać się postępów, a nie regresu w nauczaniu.



Rys. 3. Prognozowanie progu osiągalności umiejętności programistycznych, nabywanych w szkoleniach etapowych, dla $w_0=0,1$

Fig. 3. Prediction of achievement's threshold of programming skills obtained during the stage-based training, for $w_0=0,1$

Istotnym w tej sytuacji wydaje się zadanie ustalenia stosunku wzajemnego pomiędzy skalą umiejętności $u_i \in [0...1]$ i skalą ocen stosowaną na uczelni $O_i \in [2, 3, 3+, 4, 4+, 5]$. Jednym z możliwych rozwiązań może być wykorzystanie werbalno-liczbowej skali Johna Harringtona [4], w której poprzez funkcję optymalnego zróżnicowania ustalone są progi zgodne z tabelą 1.

Tab. 1. Werbalno-liczbowe skale Harringtona w kontekście testowania umiejętności

Tab. 1. Verbal numerical scale of Harrington in the test of ability context

Poziom jakości	Współczynnik Harringtona	Ocena konwencjonalna
Bardzo wysoki / Wysoki	0,80...1,00	Bardzo dobry / Dobry plus
Powyżej średniego	0,64...0,80	Dobry
Średni	0,37...0,64	Dostateczny plus
Niski	0,20...0,37	Dostateczny
Bardzo niski	0,00...0,20	Niedostateczny

Wyraźna nieliniowość prognozowanych osiągnięć powoduje to, że istnieje minimalny poziom nabytych umiejętności przy którym ma sens przystępować do ćwiczeń na kolejnym etapie. Np., żeby osiągnąć ocenę na poziomie (i+1) „dostateczny” $u_{i+1} \geq 0,20$ w złożonym szkoleniu ($k=4, j=4$) wymagane są umiejętności z etapu (i) minimum na poziomie „dostateczny plus” $u_i \geq 0,42$. Jak widać z wykresu, zakres umiejętności $0 \leq u_i \leq u_{imin}$, który nie daje szans studentowi na opanowanie nowych umiejętności u_{i+1} na przyzwoitym poziomie rozszerza się ze wzrostem komplikacji i wariantowości kursu.

Zestawienie pięciostopniowej skali jakości Harringtona z sześciostopniową skalą ocen

konwencjonalnych wymaga bardziej szczegółowych rozważań, na potrzeby jakościowej analizy wyników modelowania przyjęliśmy, że oceny będą przeliczane jak pokazano w tabeli 1. Dało to możliwość naniesienia na wykres prawdopodobieństwa „obszarów” ocen, co pozwala analizować sugerowany model w terminologii skali werbalnej. Jak można zauważyć, np. na polepszenie ocen na kolejnych poziomach szkolenia mają szansę raczej ci studenci, którzy mają wysokie oceny z poprzedniego etapu.

4 Podsumowanie

Autorzy nie uwzględnili w swoim modelu takiego poważnego czynnika, jako motywacja studenta do nauki. Wiadomo, że zmiana motywacji może istotnie zmienić postępy w szkoleniu. Jednak „mierzalność” poziomu motywacji jest zadaniem, które wychodzi za zakres naszych kompetencji.

Literatura

1. Strykowski W., Skrzydlewski W.: *Dokąd zmierza technologia kształcenia*, UAM, Poznań 1993.
2. Jaskuła B.: *Projektowanie i zastosowanie dydaktycznych systemów komputerowych*. Wydawnictwo FOSZE, Rzeszów, 1995.
3. Lindsay P., Norman D.: *Procesy przetwarzania informacji u człowieka*, PWN, Warszawa, 1991.
4. Larichev O.I., Moshkovich H.M.: *Verbal Decision Analysis for Unstructured Problems*, Kluwer Academic Publishers, Boston 1997.

Streszczenie

Profesjonalne programowanie komputerów wymaga opanowania zbioru określonej wiedzy i umiejętności, do weryfikacji których można posłużyć się techniką testowania skomputeryzowanego. Przedstawiona w artykule koncepcja etapowego testowania zakłada budowę algorytmu testującego w oparciu o modelowanie prawdopodobieństwa utrwalania nowych umiejętności na podstawie poprzednich osiągnięć studenta. Algorytm uwzględnia tak zdolność studenta do nabycia wiedzy, jak i stopień komplikacji kursu. Do wystawiania oceny sugerowane jest zastosowanie obiektywnej skali werbalno-liczbowej.

Idea of automated verification of computer's programming knowledge and skills

Summary

Professional programming of computers requires package of definite knowledge and abilities that can be verified by computerized testing technique. The concept of stage testing, which is presented in the article sets up the structure of algorithm based on modeling credibility of new level of ability on base of former achievements of student. Algorithm takes into consideration ability of student for procurement of knowledge, as well as degree of complication of discipline. Application of an objective verbal-numerical scale is suggested for exposure of assessment.