This paper should be cited as:

K. Barteczko, "Założenia Systemu Doskonalenia Kwalifikacji Programistycznych w ramach zdalnego nauczania," in Postępy e-edukacji, L. Banachowski, Ed. Warszawa: Wydawnictwo PJWSTK, 2013, pp. 107–116.

Rozdział 6

Założenia Systemu Doskonalenia Kwalifikacji Programistycznych w ramach zdalnego nauczania

Krzysztof Barteczko

Polsko-Japońska Wyższa Szkoła Technik Komputerowych

Metody i sposoby nauczanie programowania od lat są przedmiotem dyskusji i prac naukowych (por. [1], [2], [3], [4]). Od pewnego czasu dydaktyka zyskuje także praktyczne wsparcie w systemach automatycznej oceny rozwiązań zadań programistycznych. W Polsko-Japońskiej Wyższej Szkole Technik Komputerowych przygotowano założenia Systemu Doskonalenia Kwalifikacji Programistycznych, realizowanego w ramach projektu „Informatyk – wszechstronny specjalista", finansowanego z Programu Operacyjnego Kapitał Ludzki. Głównym celem Systemu Doskonalenia Kwalifikacji Programistycznych jest stworzenie modelu i narzędzi pogłębionej analizy i oceny postępów studentów, a także zrozumiałego, wspomagającego proces uczenia się, przedstawiania im wyników tej oceny. Realizacja tego zamierzenia będzie opierać się w głównej mierze na specjalnie przygotowanych narzędziach weryfikacji postępów studentów. Wśród znanych dotąd systemów automatycznej oceny (zob. ich przegląd w [3]), proponowane rozwiązanie wyróżnia się szerokim zakresem rodzajów wykonywanych testów, elastycznymi sposobami ich formułowania oraz koncepcją organizacji całego systemu przygotowania zadań oraz oceny ich rozwiązań.

Motywacja

Motywacja dla stworzenia Systemu Doskonalenia Kwalifikacji Programistycznych wynikła m.in. z:

- potrzeby uwzględnienia nowych trendów technologicznych i wymagań rynku pracy w wykładach i ćwiczeniach dla przedmiotów programistycznych,
- potrzeby zwiększenia stopnia badawczego nastawienia procesu kształcenia,
- potrzeby silniejszego ukierunkowania nauki przedmiotów programistycznych na praktyczne umiejętności rozwiązywania problemów programistycznych,
- potrzeby lepszego zdefiniowania celów kształcenia i przygotowywania zadań lepiej do tych celów dostosowanych,

- potrzeby poszerzenia i ujednolicenia przestrzeni kryteriów oceny postępów studentów w celu zwiększenia jakości kształcenia w ramach przedmiotów programistycznych,
- potrzeby przekształcenia sposobów oceny postępów studentów w bardziej jednolity, spójny i interaktywny model, wspierający doskonalenia ich umiejętności programistycznych,
- potrzeby stworzenia (i/lub doskonalenia) utrwalonych i powtarzalnych procedur weryfikacji osiągania efektów kształcenia,
- potrzeby lepszego monitorowania i analizy postępów studentów w celu indywidualizacji procesu nauczania w ramach przedmiotów programistycznych,
- potrzeby wsparcia pracy nauczycieli akademickich oprzyrządowaniem ułatwiającym pogłębioną ocenę, monitoring i analizę postępów studentów w zakresie przedmiotów programistycznych.

Koncepcja systemu

System będzie obejmował przedmioty programistyczne, wykorzystujące technologie Javy. Ważnym elementem jest tu ukierunkowanie na doskonalenie praktycznych umiejętności rozwiązywania problemów programistycznych. Dlatego centralne miejsce w prowadzonych pracach zajmuje aspekt przygotowania zadań programistycznych oraz oceny wyników studentów. W tym celu budowany jest zestaw narzędzi informatycznych, umożliwiający pogłębioną analizę i oceną rozwiązań zadań programistycznych oraz prezentację wyników tej oceny studentom. Zostanie również przeprowadzona aktualizacja treści dydaktycznych oraz przygotowana baza zadań w formie wymaganej przez system.

Można zaproponować następującą listę umiejętności, które należałoby kształcić w trakcie nauki programowania [1]:

1. Twórcze projektowania rozwiązań problemów.
2. Umiejętność doboru i stosowania właściwych konstrukcji językowych,
3. Umiejętność doboru i stosowania właściwych środków (bibliotek, szkieletów, wzorców projektowych).
4. Trzymanie się specyfikacji i spełnianie wymagań.
5. Pisanie poprawnego kodu z uwzględnieniem granicznych przypadków.
6. Pisanie kodu dobrej jakości (czytelność, zgodność z zasadami dobrego programowania),
7. Pisanie uniwersalnego i elastycznego kodu:
   a. słabych powiązaniach pomiędzy różnymi częściami (loose coupling),
   b. kodu przygotowanego na zmiany wymagań i funkcjonalności,
   c. skalowalnego.
8. Pisanie kodu efektywnego.

Te osiem punktów-celów nauczania programowania musi znaleźć odbicie w różnych rodzajach treści dydaktycznych, w szczególności zadań przygotowywanych dla studentów. Treści zadań muszą być również przygotowywane z uwzględnieniem aspektów praktycznych (m.in. wymagań rynku pracy), technologicznego (nowe trendy), badawczych (zadania twórcze wymagające własnych analiz i badań studentów).

Nie każde zadanie w równym stopniu będzie realizowało każdy z wymienionych punktów-celów czy aspektów, nie w każdej fazie nauki każdy z tych punktów będzie tak samo istotny. Ale w całym zestawie kursów programistycznych należy dążyć do uwzględnienia wszystkich przedstawionych celów i aspektów.

Kryteria oceny rozwiązań zadań wynikają bezpośrednio z przedstawionych ośmiu celów nauczania. Taka kompleksowa ocena nie jest łatwa nawet dla pojedynczego zadania. Dla prowadzącego, mającego pod opieką wielu studentów jest szczególnie trudna. Wymaga zatem zastosowania dodatkowych narzędzi oraz – przynajmniej częściowej – automatyzacji procesu oceny i indywidualnego komentowania poszczególnych rozwiązań. Informacja o wynikach różnych testów, o tym gdzie i jakie błędy zostały popełnione, winna być udostępniana studentom z możliwością poprawienia rozwiązań, co da prowadzącemu możliwość lepszego monitorowania procesu dydaktycznego (gdzie są największe trudności? jakie są postępy poszczególnych studentów?).

Struktura systemu

Głównymi składowymi systemu będą:

- moduły narzędziowe (podsystemy informatyczne), służące do wsparcia sprawdzania rozwiązań zadań programistycznych i oceny postępów studentów (zob. tabela 6.1),
- podsystem zadań programistycznych.

Tabela 6.1. Podsystemy (moduły) automatycznej weryfikacji i oceny rozwiązań

Moduł	Podstawowa funkcjonalność	Uwagi realizacyjne
Moduł antyplagiatowy	Automatyczne wykrycie niebudzących wątpliwości przypadków powielania cudzych kodów i odrzucenie takich rozwiązań; identyfikacja potencjalnych plagiatów do dalszej jakościowej analizy.	Zostaną opracowane oryginalne algorytmy identyfikacji powielonych kodów dla prostych niewielkich zadań oraz oprogramowane zmodyfikowane znane algorytmy wykrywania podobieństw w kodach z uwzględnieniem odpowiedniego grupowania dużej liczby złożonych strukturalnie projektów studenckich.
Moduł weryfikacji poprawności rozwiązań	Automatyczne sprawdzanie poprawności rozwiązań ze względu na specyfikację zadania i/lub specyfikacje wejścia-wyjścia.	Narzędzia testowania poprawności, będą uwzględniać także bardziej skomplikowane przypadki (programowanie GUI, web-aplikacje, programowanie współbieżne).

Moduł	Podstawowa funkcjonalność	Uwagi realizacyjne
Moduł weryfikacji doboru środków i spełniania wymagań	Automatyczna weryfikacja doboru i stosowania właściwych konstrukcji językowych, bibliotek; spełniania wymagań formalnych postawionych w zadaniu.	Będą przygotowane proste konfigurowalne narzędzia analityczne, sprawdzające właściwe zastosowanie podejść i konstrukcji językowych oraz bibliotek zewnętrznych.
Moduł weryfikacji jakości i stylu programowania	Automatyczna weryfikacja kodu pod względem zgodności z zasadami dobrego programowania.	Zostanie zastosowany zestaw narzędzi, m.in. weryfikujący złożoność, czytelność, styl kodu.
Moduł testowania efektywności rozwiązań	Automatyczne weryfikacja efektywności programów.	Jako specjalny przypadek testów poprawności rozwiązań.
Moduł testowania skalowalności, elastyczności i uniwersalności	Automatyczne sprawdzanie zachowania programów w warunkach dużego obciążenia i zmiany wymagań.	Jako specjalny przypadek testów poprawności i efektywności rozwiązań.
Wszystkie wymienione moduły winny generować: • informacje analityczne (wskazania błędów w rozwiązaniach) w formie zrozumiałej dla wykładowców i studentów, • punktowe propozycje oceny rozwiązania. W tworzeniu modułów zostaną zastosowane, oprócz oprogramowanych oryginalnych algorytmów, wybrane biblioteki (pakiety) typu „open source”. Będą one integrowane z ww. modułami narzędziowymi.		

Podsystem zadań programistycznych będzie obejmował:

- bazę zadań,
- procedury przygotowania zadań zgodnie z wymaganiami systemu (różne rodzaje zadań będą wymagały odpowiedniego przygotowania, tak by np. mogła być automatycznie weryfikowana poprawność rozwiązań),
- interfejsy i narzędzia umożliwiające:
  - dla dydaktyków:
    - dodawanie zadań do bazy,
    - generowanie indywidualnych komentarzy i ocenę rozwiązań studentów z uwzględnieniem wyników z podsystemu modułów automatycznej weryfikacji,
  - dla studentów:
    - dostęp do zadań i przygotowywanie ich rozwiązań w odpowiedniej formie,
    - dostęp do komentowanych wyników oceny.

Zasady budowy i schemat działania systemu

Budowa i działanie systemu będą oparte na trzech zasadach:

- minimum administracji,
- „uczyć, a nie karać”,
- maksimum autonomii dydaktyków (prowadzących zajęcia ze studentami).

W takcie prac nad oprogramowaniem i wdrożeniem systemu zostanie stworzona systemowa baza zadań – SYSTASKS.

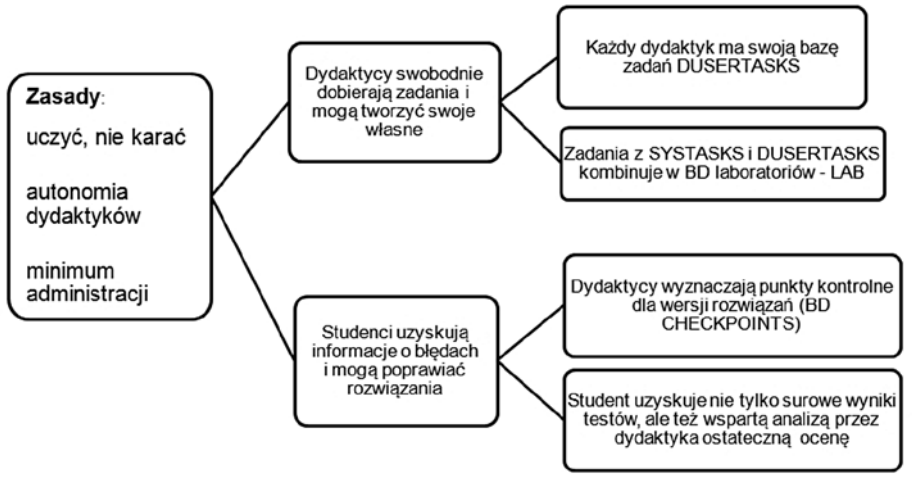
Oprócz tego każdy dydaktyk (DUSER) będzie mógł przygotowywać i prowadzić własną bazę zadań (USERTASKS) i na zajęciach kombinować zadania z SYSTASKS i USERTASKS.

Jednostką pracy ze studentami będą laboratoria (LAB), czyli przygotowane przez dydaktyków dowolne połączenia zestawów zadań z baz SYSTASKS i USERTASKS.

Przygotowany przez dydaktyka LAB jest udostępniane studentom, studenci wykonują zadania z zestawu i przekazują rozwiązanie w postaci projektu (PROJ).

Projekty podlegają automatycznemu sprawdzeniu przez narzędzia systemu, w tym zakresie, w jakim określają to definicje zadań zestawu LAB, a wyniki automatycznego sprawdzenia są udostępniane studentom po uprzedniej weryfikacji przez dydaktyka i ew. korektach/komentarzach. Studenci mają następnie możliwość poprawienia rozwiązań i przekazania ich w postaci kolejnych wersji projektu, który też podlega automatycznej weryfikacji oraz pogłębionej ocenie dydaktyka. Harmonogram i możliwości poprawiania rozwiązań określa dydaktyk w swojej bazie CHECKPOINT (określającej liczbę i terminy sprawdzeń kolejnych rozwiązań).

Rysunek 6.1 przedstawia odzwierciedlenie przyjętych zasad budowy systemu w sposobie jego działania.



Rysunek 6.1. Zasady budowy i działania systemu

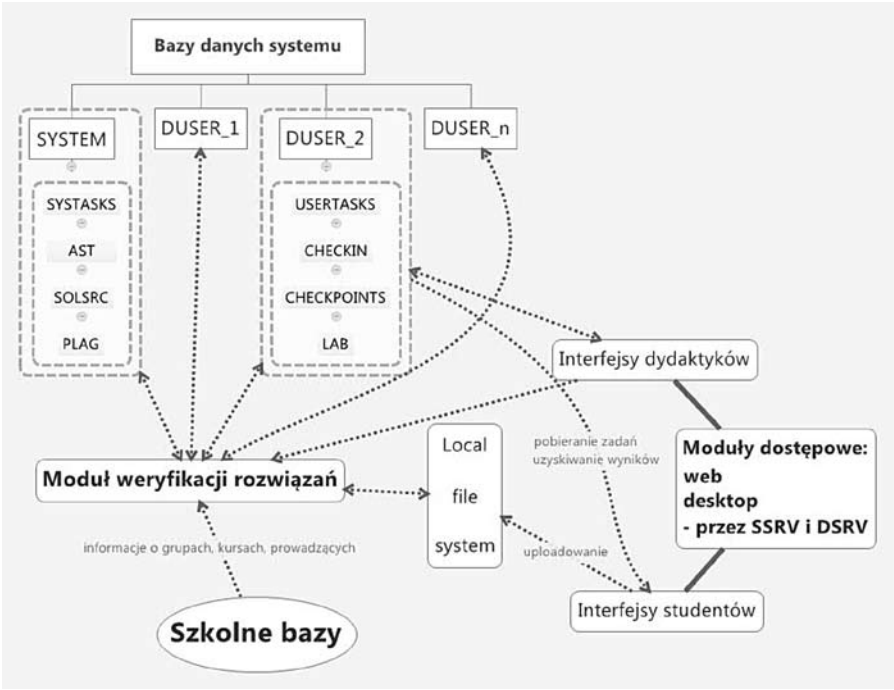
Sumaryczna ocena wyniku studenta dla danego LAB jest formułowana na podstawie punktacji za obie wersje rozwiązań.

Wyniki automatycznego sprawdzania każdego LAB (dla każdej wersji rozwiązań) będą zapisywane do bazy danych postępów studentów (CHECKIN) w postaci rekordów dla każdej wersji rozwiązania każdego zadania z zestawu LAB.

Oprócz tego wszystkie rozwiązania studentów będą podlegać podstawowym testom antyplagiatowym, których wyniki będą zapisywane w bazie PLAG, dla której wsparciem będa baza kodów źródłowych rozwiązań (SOLSRC) oraz baza AST kodów (AST).

Dostęp do usług systemu będzie realizowany poprzez interfejsy webowe oraz aplikacje desktopowe, komunikujące się z systemem za pośrednictwem serwerów, odrębnych dla dydaktyków (DSERV) oraz studentów (SSERV).

Zarys całościowej budowy systemu przedstawiono na rysunku 6.2.



Rysunek 6.2. Zarys całościowej budowy systemu

O procedurach weryfikacji rozwiązań

Zgodnie z prezentacją w tablicy 1, system ma:

- testować poprawność rozwiązań,
- weryfikować czy rozwiązanie spełnia postawione wymagania,
- sprawdzać jakość kodu, jego efektywność i skalowalność.

Automatyczna weryfikacja poprawności programów nie jest zadaniem łatwym, a w ogólnym przypadku jest to na razie zadanie niewykonalne. W systemie przyjęto

więc założenie, że rozwiązania będą *testowane*, co oczywiście pozwala wykryć jakiś zakres błędów, ale nie dowodzi, że program jest poprawny w każdym przypadku.

Automatyczne testowanie poprawności działania możliwe jest dla następujących rodzajów zadań programistycznych (zob. [1]):

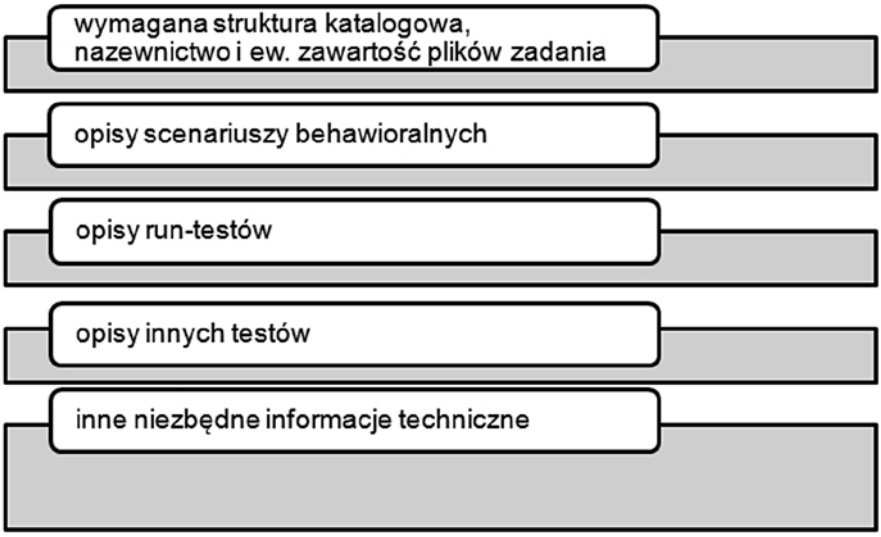
A. „DOPASOWANIE DO KODU": dany jest gotowy fragment uruchomieniowy, wyprowadzający wyniki na konsolę lub do pliku, a zadanie polega na dopisaniu brakujących fragmentów (klas, metod),
B. „CZARNA SKRZYNKA": zadanie polega na wczytaniu danych wejściowych i zapisaniu wyników (wejście: pliki, bazy danych, wyjście: pliki, bazy, konsola),
C. „PO SPECYFIKACJI": zadanie zawiera dość szczegółową specyfikację (jakie klasy, metody itp. należy zdefiniować), a testowanie polega na sprawdzeniu zgodności implementacji z wymaganiami.

W przypadkach A i B programy studentów będą automatycznie uruchamiane, a wyniki ich działania sprawdzane przez procedury testujące, zapisywane w specjalnej formie (języku). Ten rodzaj testów nazwiemy run-testami. W przypadku C mamy do czynienia z testami behawioralnymi, czyli automatycznym sprawdzaniem czy poszczególne elementy programu (takie jak np. metody jakiejś klasy) zachowują się zgodnie z wymaganiami. Tutaj stosowane będą narzędzia testowania behawioralnego z dodatkowym wsparciem dla opisu testu i jego wyników. We wszystkich rodzajach testowania poprawności system będzie działał w bezpiecznym trybie („sandbox"), uniemożliwiającym programom studenckim wykonywanie niedozwolonych działań.

Do weryfikacji spełniania wymagań oraz jakości kodu zastosowana będzie statyczna analiza kodu (z użyciem regularnych wyrażeń oraz gotowych, darmowych, środowisk statycznej analizy kodu). Dodatkowo – wraz z odpowiednio konfigurowanymi run-testami umożliwi to (częściową) analizę rozwiązań pod kątem efektywności i skalowalności.

Weryfikacja rozwiązań studentów wymaga przygotowania odpowiednich definicji zadań. Schematycznie przedstawia to rysunek 6.3.

Widoczne na rysunku 6.3 wymagania na strukturę katalogową, nazewnictwo plików, a nawet ich (częściową) zawartość są niezbędne dla wielu rodzajów testów (jeśli ich wyniki mają być wiarygodne). Spełnienie tych wymagań musi być sprawdzane przed uruchomieniem testów. Zajmie się tym specjalny moduł Projector. Na tym jednak nie kończy się jego rola. Doświadczenie dowodzi, że wymagań na strukturę katalogową, nazewnictwa plików itp., nie wystarczy podać w tekście zadania, bowiem nie zawsze są one respektowane przez studentów (co może wynikać z niezrozumienia, pośpiechu itp.). Dlatego Projector zajmuje się również generowaniem struktur katalogowych i zawartości projektów dla studentów, a także umożliwia im sprawdzenie formalnej poprawności projektu przed jego przekazaniem do oceny. Zasada „nie karać, ale uczyć" odgrywa tu ważną rolę: niespełnienie formalnych wymogów co do struktury projektu skutkuje odrzuceniem rozwiązania ze szkodą dla oceny jego innych elementów (która to ocena ma posłużyć poprawie rozwiązania).



Rys. 6.3. Definicje zadań

Obrazowo ideę działania Projectora ilustruje rysunek 6.4.

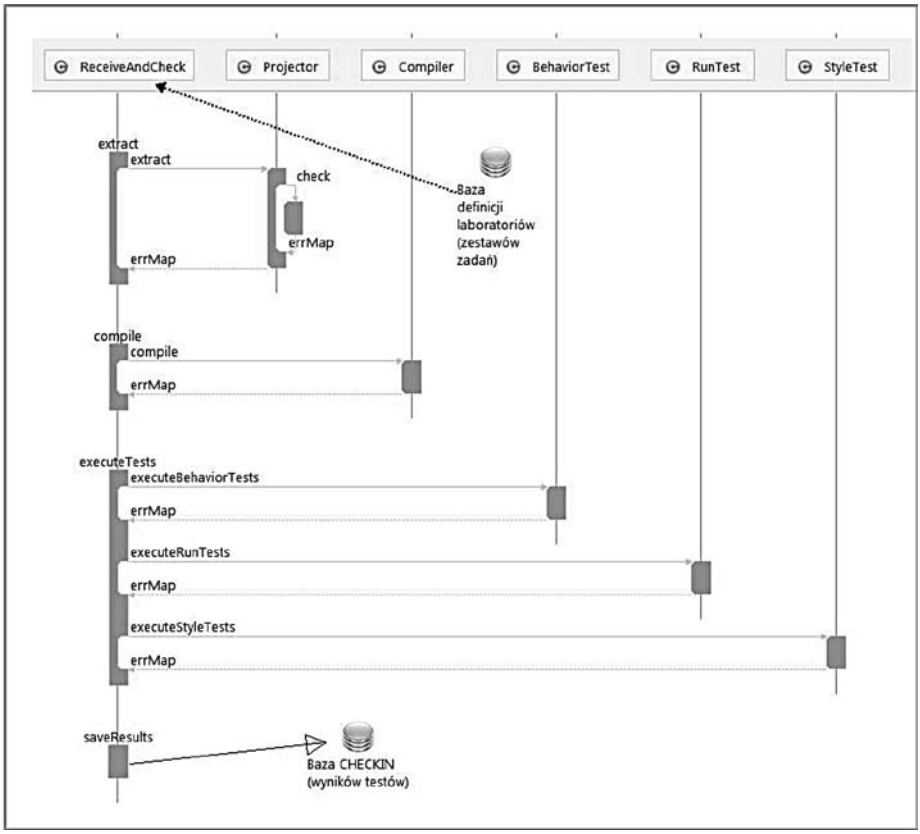
Rysunek 6.4. Projector pomaga spełniać niezbędne wymogi formalne rozwiązań

Procedura weryfikacji rozwiązań składać się będzie z następujących kroków:

1. Projector pobiera skompresowane projekty, sprawdza poprawność ich poszczególnych składowych i jeśli projekt jest poprawny (to znaczy co najmniej jeden z jego elementów jest poprawny), rozpakowuje go do przestrzeni DUSERA. Wszelkie informacje o błędach są rejestrowane w bazie CHECKIN.
2. Compiler kompiluje projekty. Informacje o błędach w kompilacji zapisywane są w bazie CHECKIN.
3. Pobranie z definicji zadań informacji o testach do wykonania (dla różnych zadań mogą być wykonywane różne testy) uruchamia odpowiednie rodzaje te-

stów (RunTest, BehaviorTest, StyleTest itd.). Wyniki są zapisywane do bazy CHECKIN.

Uproszczony schemat weryfikacji rozwiązań ilustruje rysunek 6.5.



Rysunek 6.5. Uproszczony schemat testowania

Kontekst zdalnego nauczania

System może być szczególnie użyteczny w zdalnej edukacji, ponieważ:

- znacząco ułatwi dydaktykom sprawdzanie rozwiązań studentów i monitorowanie ich postępów,
- zapewni studentom informację o błędach i sposobach ich unikania w postaci trwałej (na platformie elektronicznej), a nie tylko na zasadzie „luźnej" rozmowy z dydaktykiem na zajęciach.

Planowana jest integracja systemu z platformą zdalnego nauczania PJWSTK – EDUX, co oznacza, że:

- system będzie łatwo dostępny w ramach e-kursów,
- stacjonarne zajęcia programistyczne zyskają dodatkowy wymiar zdalnego nauczania (np. zadania wykonywane przez studentów w domu mogą być automatycznie sprawdzane, a studenci będą mogli poprawiać rozwiązania jeszcze przed ostateczną oceną dydaktyka na ćwiczeniach w Szkole).

Literatura

1. Barteczko K., Zadania programistyczne w zdalnym nauczaniu, w: EduAkcja, Magazyn Edukacji Elektronicznej, 1 (3), 2012.
2. Bennedsen J, Caspersen M. E., Kölling M., Reflections on the Teaching of Programming: Methods and Implementations, Springer 2008.
3. Bhantola, P. et al. Review of Recent Systems for Automatic Assessment of Programming Assignments, Koli Calling 2010.
4. Robins A., Rountree J., Rountree N., Learning and Teaching Programming: A Review and Discussion, Computer Science Education 2003, Vol. 13, No. 2.

This paper should be cited as:

K. Barteczko, “Założenia Systemu Doskonalenia Klasyfikacji Programistycznych w ramach zdalnego nauczania,” in Postępy e-edukacji, L. Banachowski, Ed. Warszawa: Wydawnictwo PJWSTK, 2013, pp. 107–116.