JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 8 (2018) 270-276
Wysłane: 2018-06-23
Przyjęte: 2018-06-27

Metody weryfikujące poziom wiedzy i umiejętności programisty

Paweł Hajduk*, Norbert Wieruszewski*, Maria Skublewska-Paszkowska
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Artykuł opisuje aktualnie stosowane metody weryfikacji poziomu wiedzy i umiejętności programistów. Do realizacji badań wykorzystano własne rozwiązanie w postaci aplikacji implementującej kilka wybranych metod, na której następnie przeprowadzono testy użytkowe przy udziale programistów o zróżnicowanym poziomie doświadczenia, wiedzy i umiejętności. Na podstawie analizy uzyskanych wyników wyciągnięto wnioski, które pozwoliły na ocenienie każdej z metod w następujących kategoriach: skuteczność sprawdzenia użytkownika, niezawodność działania metody, czas weryfikacji rozwiązania, atrakcyjność użytkowania oraz uniwersalność metody.

Słowa kluczowe: automatyczna ocena programistów; metody weryfikacji wiedzy; testy jednostkowe; analiza statyczna kodu

*Autor do korespondencji.

Adresy e-mail: pawel.hajduk@pollub.edu.pl, norbert.wieruszewski@pollub.edu.pl

Verification methods of a programmer's knowledge and skills

Paweł Hajduk*, Norbert Wieruszewski*, Maria Skublewska-Paszkowska
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The article describes currently utilized methods of a programmer's knowledge verification and skills. The research consisted of creating custom solution which was an application implementing chosen methods and carrying out test with the participation of programmers having various levels of experience, knowledge and skills. Effectiveness of assessment, reliability and verification time were evaluated based on an analysis of the results received from the research.

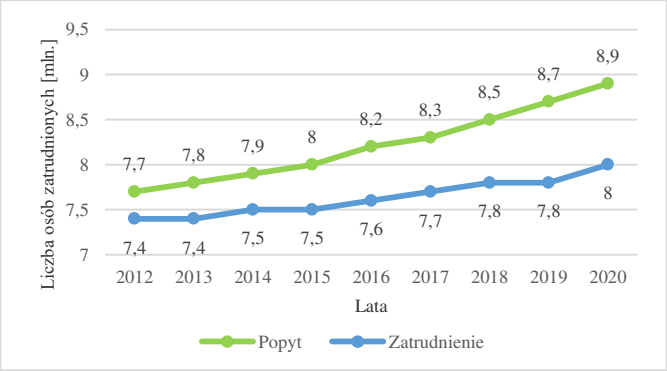
Keywords: automatic programmers assessment; knowledge verification methods; unit tests; static code analysis

*Corresponding author.

E-mail addresses: pawel.hajduk@pollub.edu.pl, norbert.wieruszewski@pollub.edu.pl

1. Wstęp

XXI wiek to dalszy ciąg okresu dynamicznego rozwoju branży informatycznej. Wynikiem tego jest fakt, że ogólnoświatowe zapotrzebowanie na specjalistów IT nie przestaje rosnąć. Jest to przede wszystkim efekt działań dotyczących pełnej informatyzacji kolejnych przedsiębiorstw w różnych gałęziach gospodarki. Rys. 1 przedstawia wykres wzrostu zatrudnienia w branży IT, z którego łatwo jest odczytać, że popyt znacznie przewyższa zatrudnienie.



Rys. 1. Wzrost zatrudnienia w branży ICT w krajach Unii Europejskiej (w mln) [1]

Firmy z branży IT stają przed problemem polegającym na zaspokojeniu wysokiego popytu na wykwalifikowanych i doświadczonych informatyków. Na rynku nie brakuje chętnych do pracy kandydatów, jednak znalezienie osoby o wymaganych kwalifikacjach, w założonym wcześniej czasie, jest bardzo trudne. Z uwagi na ogromną liczbę kandydatów, których tylko niewielki procent spełnia wymagania pracodawcy na dane stanowisko, a także na czas jaki musi poświęcić pojedyncza osoba techniczna na zweryfikowanie wiedzy i sprawdzenie testu praktycznego każdego z kandydatów, proces rekrutacji programistów wydaje się być dobrym przykładem procesu, który należałoby usprawnić i w jak najwyższym stopniu zautomatyzować [2].

Celem badania było przeprowadzenie analizy metod oceniających poziom wiedzy programistycznej, wykrycie ich wad oraz zaproponowanie alternatywnych rozwiązań. Na podstawie uzyskanych wyników metody zostały przeanalizowane w celu odnalezienia ich słabych punktów, które powinny zostać w przyszłości wyeliminowane.

Celem pośrednim była implementacja aplikacji pozwalającej na przeprowadzenie testów użytkowych z wykorzystaniem wybranych metod weryfikacji poziomu wiedzy programistycznej.

2. Przegląd literatury

W celu wprowadzenia automatyzacji do analizowanego procesu należy najpierw wydzielić z niego te fragmenty, które automatyzacji wymagają, a zarazem, dla których jest ona w ogóle możliwa. Do takich fragmentów z pewnością można

zaliczyć etap weryfikacji poprawności i jakości kodu programu. Poprawność składni, czyli zgodność kodu z gramatyką i zasadami danego języka, jest poddawana analizie między innymi na etapie kompilacji przeprowadzanej przez kompilatory, które posiadają algorytmy zaimplementowane specjalnie w tym właśnie celu. Poprawność logiczna natomiast, która ma największe znaczenie w kontekście oceny umiejętności badanej osoby, określa to, czy dany program działa prawidłowo z różnymi zadanymi wartościami wejściowymi. Tego typu weryfikacja może zostać przeprowadzona przy użyciu mechanizmu testów jednostkowych.

Prawidłowy kod programu nie zawsze charakteryzuje się wysoką jakością, co oznacza, że jej poziom powinien zostać poddany oddzielnej analizie. Przykładem takiego sprawdzenia jest analiza statyczna kodu badająca kod źródłowy programu bez jego uruchamiania w wyniku czego możliwe jest wyznaczenie parametrów takich jak złożoność obliczeniowa.

2.1. Kompilacja dowodem poprawności składni kodu programu

W poszukiwaniu rozwiązania mającego posłużyć za swego rodzaju weryfikator poprawności składni kodu rozważyć można wykorzystanie gotowych i w założeniu niezawodnych kompilatorów. Głównym zadaniem tych aplikacji jest przepisanie kodu programu wejściowego, napisanego najczęściej w języku wysokiego poziomu, na kod programu wyjściowego innego języka [3]. W czasie skomplikowanego procesu kompilacji przeprowadzane zostają między innymi trzy analizy: leksykalna, składniowa i semantyczna [4].

Analiza leksykalna dzieli kod wejściowy na pojedyncze jednostki leksykalne opisane w gramatyce danego języka programowania - tak zwane leksemy. Jeżeli kompilator znajdzie wyrażenie, które nie zostało opisane w gramatyce, to zostanie ono zgłoszone jako błąd leksykalny.

W fazie analizy składniowej sprawdzeniu podlegają wydzielone wcześniej tokeny, a dokładniej ich wzajemne ustawienia. Dobrym przykładem mogą być nawiasy klamrowe, które otaczają bloki kodu. Każdy otwarty nawias klamrowy powinien również zostać zamknięty. W przeciwnym wypadku taka sytuacja zostanie zgłoszona jako błąd składniowy [5].

Jako ostatnia z analiz w procesie kompilacji zostaje przeprowadzona analiza semantyczna. Jej zadaniem jest sprawdzenie kodu programu w celu wykrycia błędów wynikających z nieprawidłowego użycia elementów języka, które same w sobie można by uznać za poprawne.

2.2. Poprawność logiczna programu potwierdzona testami jednostkowymi

Testy jednostkowe to sposób na sprawdzanie działania najmniejszych części programu, które można logicznie odseparować od reszty. Przyjmuje się, że testowany fragment programu wykonuje pojedynczą jednostkę pracy, niekoniecznie jest to odrębna część kodu np. metoda lub klasa [6]. W procesie wytwarzania oprogramowania testy jednostkowe są jednym z ważniejszych wyznaczników poprawności działania systemów informatycznych, a dzięki popularności tej metody, istnieje wiele narzędzi pozwalających pisać i wykonywać testy jednostkowe. Odpowiednio dobrany zestaw testów jednostkowych może wykryć wiele przypadków, gdzie wykonywany kod nie realizuje swojego zadania [7].

2.3. Analiza statyczna kodu

Sprawdzenie poziomu skomplikowania programu to bardzo rozległe zagadnienie, ale na potrzeby stosowanych metod, wystarczającym okazało się zawężenie pojęcia złożoności kodu wyłącznie do następujących aspektów: łatwości zrozumienia kodu, testowalności kodu, łatwości utrzymania kodu [8]. Istnieje wiele różnych metod, którymi można wyznaczyć poziom złożoności kodu, ale na potrzeby niniejszego artykułu postanowiono opisać jedynie dwie z nich: obliczanie złożoności cyklicznej oraz złożoności poznawczej.

2.3.1. Złożoność cykliczna

Złożoność programu można ocenić na kilka sposobów. Jednym z nich jest obliczenie złożoności cyklicznej, która zlicza liczbę możliwych rozgałęzień scenariusza odtworzonego w programie przy pomocy grafu przepływu sterowania, czyli liczbę przypadków testowych koniecznych do wykonania w celu pełnego pokrycia funkcjonalności kodu. Niestety metoda ta posiada pewne niedoskonałości, np. jest nastawiona na sprawdzanie poziomu skomplikowania programu z punktu widzenia wykonującej go maszyny. Z tego względu można zauważyć pewne przypadki, gdy złożoność cykliczna nie jest wyznacznikiem łatwości zrozumienia, testowania i utrzymywania kodu [8].

2.3.2. Złożoność poznawcza

Kolejną metodą obliczania złożoności programu jest złożoność poznawcza [8]. Opiera się ona na ocenie, jak trudny jest do zrozumienia przepływ programu zgodnie z poniższymi regułami [8]:

- jeżeli występuje przerwa w ciągłości przepływu programu, zwiększ wskaźnik złożoności;
- jeżeli struktury powodujące zmianę przepływu programu są zagnieżdżone, zwiększ wskaźnik złożoności;
- jeżeli wiele linii kodu w czytelny sposób zostało zastąpionych pojedynczą linią kodu, nie zwiększaj wskaźnika złożoności.

3. Aktualnie stosowane metody weryfikacji wiedzy i umiejętności programisty

3.1. Weryfikacja poziomu wiedzy przy użyciu zadań otwartych

Zadania otwarte to typ zadań, który wymaga od osoby badanej udzielenia swobodnej i autorskiej odpowiedzi, zazwyczaj nieco dłuższej niż jedno zdanie. Pytania typu

otwartego wymagają najczęściej wyjaśnienia znaczenia danego zagadnienia, opisania działania metody/procesu lub przedstawienia jakiegoś konkretnego zjawiska. Sprawdzenie tego typu zadania bez udziału osoby sprawdzającej może okazać się trudne do osiągnięcia - niemożliwym jest przygotowanie pełnego klucza odpowiedzi.

3.2. Weryfikacja poziomu wiedzy przy użyciu zadań zamkniętych

Zadania zamknięte lub inaczej zadania wyboru wymagają udzielenia odpowiedzi poprzez wytypowanie jednej lub więcej z kilku możliwości przygotowanych wcześniej przez autora zadania. Dla pytań tego typu przygotowanie mechanizmu automatycznego sprawdzania nie stanowi większego problemu - oznaczenie prawidłowych odpowiedzi w definicji zadania i przyjęcie jednej z kilku możliwych zasad punktowania zadania wystarczą do zweryfikowania i ocenienia otrzymanych odpowiedzi.

3.3. Weryfikacja poziomu umiejętności przy użyciu zadań programistycznych

Powyższa kategoria zadań opiera się na modyfikacji kodu programu w celu osiągnięcia wskazanych w treści polecenia założeń. Wśród zadań programistycznych zostały wyróżnione następujące typy.

3.3.1. Zadania algorytmiczne

Zadania algorytmiczne polegają na zaimplementowaniu uniwersalnego i optymalnego rozwiązania problemu podanego w treści zadania. Oceniana jest poprawność działania algorytmu, czas wykonania programu, ale również czytelność i łatwość zrozumienia kodu. Rozwiązanie powinno zaliczyć wszystkie zdefiniowane przypadki testowe, składające się ze zwyczajnych, ale również wyjątkowych scenariuszy, pozwalających wykryć kompletność rozwiązania oraz zabezpieczenie go przed nietypowymi danymi wejściowymi [9].

3.3.2. Poprawienie błędów w kodzie

Umiejętność wprowadzania poprawek do istniejącego kodu jest również bardzo pożądana u dobrego programisty. Wspomniana czynność często stanowi większość czasu poświęconego na rozwijanie nowych funkcjonalności w oprogramowaniu. Zadanie z naprawianiem błędów polega na przeczytaniu i zrozumieniu kodu, zidentyfikowaniu jakie błędy w nim występują, a następnie wyeliminowaniu odnalezionych defektów. Poprawność działania programu jest weryfikowana uruchomieniem zestawu testów jednostkowych i analizą statyczną kodu.

3.3.3. Poprawne wykorzystanie istniejącego kodu

Obecnie oprogramowanie wytwarzane przez firmy informatyczne jest na tyle złożone, że prace nad nim prowadzone są przez całe zespoły programistów. Zdolność wykorzystania kodu napisanego przez innych jest cenna, gdyż pozwala zminimalizować czas poświęcony na rozwijanie i testowanie kodu. Poziom opanowania wspomnianej umiejętności jest weryfikowany przez przygotowanie metod, które następnie mają być wykorzystane w celu napisania programu realizującego podane w poleceniu cele. Poprawność działania programu jest sprawdzana testami jednostkowymi.

3.3.4. Przegląd kodu

Podobnie, jak w przypadku poprzednio opisanej metody, wykonywanie przeglądu kodu (ang. code review) jest następstwem konieczności współpracy przy wytwarzaniu oprogramowania. Wzajemne przeglądanie kodu jest dobrą praktyką pozwalającą na wczesne wykrycie wielu błędów zarówno projektowych lub architektonicznych, jak i logicznych. Sprawdzenie zdolności przeprowadzania przeglądu kodu będzie polegało na dostarczeniu działającego programu, ale w jego kodzie celowo zostaną pozostawione błędy, które będzie musiał wskazać uczestnik testu. Zadanie nie będzie obejmowało poprawy wskazanych błędów.

3.3.5. Refaktoryzacja kodu

Zakończony proces przeglądu kodu często prowadzi do potrzeby wprowadzenia poprawek. Zwykle zadania realizowane przez program nie ulegają zmianie, lecz sposób implementacji programu wymaga korekty. Na wspomnianym aspekcie wytwarzania oprogramowania koncentruje się refaktoryzacja kodu. Umiejętność przeprowadzania refaktoryzacji jest badana poprzez przedstawienie uczestnikowi testu kodu z celowo popełnionymi błędami, które będą wyraźnie zaznaczone, a zadaniem programisty będzie poprawienie ich. Końcowy sposób działania programu zostanie sprawdzony testami jednostkowymi [10].

3.3.6. Optymalizacja kodu

W zależności od zastosowania pisanych programów wydajność może być kluczowa lub mieć marginalne znaczenie, ale dobrą praktyką jest zwracanie uwagi na ograniczanie czasu wykonywania się kodu. Kandydat dostanie kod programu, który będzie napisany z pominięciem kwestii optymalizacji więc będzie wykonywał niepotrzebne lub nadmiernie złożone polecenia. Zadaniem uczestnika testu będzie wyeliminowanie defektów programu. Pod koniec nastąpi weryfikacja programu i zostanie zmierzony średni czas jego wykonania.

4. Aplikacja pomocnicza

Na potrzeby badania zaimplementowano aplikację umożliwiającą testowanie wybranych metod weryfikacji wiedzy i umiejętności programistów. Stworzona aplikacja składa się z dwóch głównych modułów:

Aplikacji Klienckiej służącej do tworzenia zadań (otwartych, zamkniętych i programistycznych), przeprowadzania testów i sprawdzania ich wyników;

Aplikacji Serwerowej odpowiedzialnej za kompilację zadań programistycznych, uruchamianie testów jednostkowych i przeprowadzanie analizy statycznej kodu.

5. Analiza wybranych metod

Metody opisane w poprzednich rozdziałach zostały poddane analizie. Do udziału w badaniu zostało wytypowanych dwanaście osób pracujących zawodowo, posiadających różne doświadczenie w zakresie programowania i zostały one podzielone na trzy równe pod względem liczebności grupy, a skład każdej z nich tworzyły: osoba po szkoleniu, osoba z rocznym stażem pracy, osoba z dwuletnim stażem pracy oraz osoba z ponad trzyletnim stażem pracy.

Przeanalizowane zostały metody polegające na rozwiązywaniu testu na kartce oraz za pomocą stworzonej aplikacji. Dodatkowo przebieg egzaminu przeprowadzanego na komputerze w jednym badaniu był nadzorowany (uczestnicy nie mogli korzystać z materiałów pomocniczych), a w drugim natomiast był przeprowadzany zdalnie, przy wykorzystaniu dowolnych pomocy, z wykluczeniem komunikacji pomiędzy uczestnikami.

Każdy test składał się z tych samych sześciu zadań: dwóch otwartych, dwóch zamkniętych i dwóch programistycznych. Na udzielenie odpowiedzi uczestnicy mieli godzinę. Pytania dotyczyły znajomości platformy Salesforce oraz podstaw algorytmiki.

5.1. Test pisemny w kontrolowanych warunkach

5.1.1. Założenia

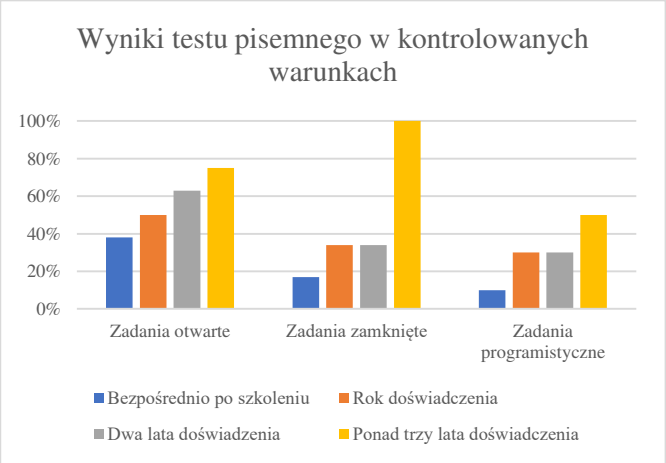
Podczas badań został zmierzony czas poświęcony na rozwiązanie poszczególnych zadań, czas spędzony na ich sprawdzenie oraz zweryfikowano, w jakim stopniu spełniają kryteria poprawności wymienione we wcześniejszej części rozdziału. W celu zredukowania wpływu obserwacji poczynań uczestników testu przez autorów pracy, zdecydowano się powierzyć obowiązek mierzenia czasu odpowiedzi na pytania samym programistom. Wiarygodność otrzymanych wyników nie była podważana przez wzgląd na profesjonalne podejście badanych osób, ale nie można wykluczyć, iż któraś z nich naruszyła ustalone zasady. Całość testu rozwiązywanego przez pojedynczego uczestnika nie powinna trwać dłużej niż godzinę.

5.1.2. Wyniki

Na Rys. 2 przedstawiono wyniki uzyskane z przeprowadzonego testu pisemnego w warunkach kontrolowanych.

5.1.3. Wnioski

Przeprowadzenie eksperymentu pozwoliło wyciągnąć liczne wnioski dotyczące skuteczności i użyteczności przeprowadzania testów pisemnych w celu weryfikacji wiedzy programistów.



Rys. 2. Wyniki testu pisemnego w kontrolowanych warunkach

Można stwierdzić, że poziom efektywności przeprowadzania testów na kartce papieru jest zadowalający dla zadań otwartych i zamkniętych. W przypadku bardziej złożonych zagadnień np. polegających na rozwiązywaniu problemów programistycznych ręczne rozwiązywanie zadań może okazać się czasochłonne i niemiarodajne. Wiele trywialnych problemów niejednokrotnie sprawiało, że programiści zamiast skupić się na rozwiązaniu problemu, zmagali się z kwestią techniczną proponowanej odpowiedzi. Takich sytuacji można było łatwo uniknąć wykorzystując powszechnie dostępne narzędzia.

Kolejnym wnioskiem, jaki może zostać wyciągnięty na podstawie wyników badania, jest potrzeba usprawnienia procesu sprawdzania zadań zawierających kod programów lub jego fragmenty zapisane odręcznie. W wymienionym przypadku utrudniona jest analiza logiki programu, ograniczona jest czytelność jego kodu, a przeprowadzanie testów dla różnych wartości wejściowych jest czasochłonne i narażone na błędy lub zaniedbanie ze względu na poziom skupienia, którego wymaga. Kwestie związane z badaniem wydajności programu dla przypadków z wieloma wartościami wejściowymi lub skomplikowanymi obliczeniami również jest niezwykle trudna do wykonania i czasochłonna.

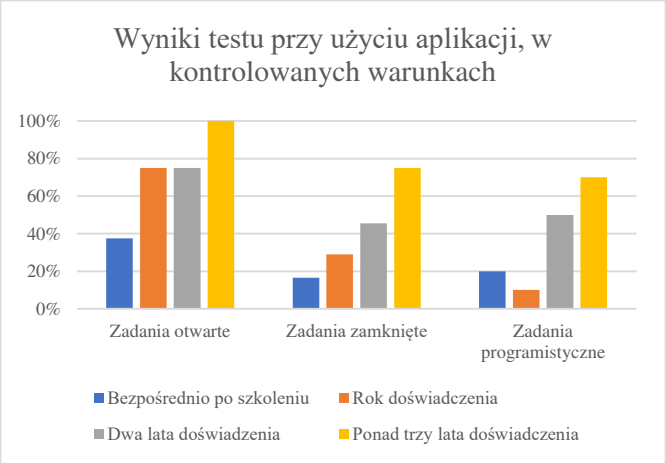
5.2. Test przeprowadzony przy użyciu stworzonej aplikacji w kontrolowanych warunkach

5.2.1. Założenia

Podczas trwania tego badania osoby rozwiązywały przygotowane zadania z wykorzystaniem aplikacji testowej. Każdy z ochotników był nadzorowany i nie miał dostępu do materiałów pomocniczych.

5.2.2. Wyniki

Na Rys. 3 przedstawiono wyniki uzyskane z przeprowadzonego testu przy użyciu stworzonej aplikacji w warunkach kontrolowanych.



Rys. 3. Wyniki testu przy użyciu aplikacji, w kontrolowanych warunkach

5.2.3. Wnioski

Na podstawie otrzymanych wyników można wysnuć kilka wniosków, które pozwalają stwierdzić, że zastosowana metoda okazała się być częściowo skuteczna. Jednym z nich jest fakt, iż znacznie lepiej widoczna jest zależność pomiędzy doświadczeniem zawodowym, a wynikami otrzymanymi w testach, co potwierdza dokładniejszą miarodajność tej metody.

Za kolejny czynnik wpływający na wiarygodność metody można uznać brak możliwości korzystania z materiałów pomocniczych, która pozwala wnioskować, że programiści posiadali niezbędną wiedzę do rozwiązania zadań. Choć brak dostępu do wiedzy na temat dziedziny rozwiązywanego problemu nie jest naturalną sytuacją w codziennej pracy programistów, to można uznać za niewątpliwą zaletę umiejętność radzenia sobie w takich okolicznościach.

Dane zgromadzone podczas wykonywanych testów pozwalają dokonać dość szczegółowej analizy prób podejmowanych przez programistów. Dzięki temu udało się stwierdzić, że możliwość kompilacji programu i zasugerowania się ewentualnymi błędami niejednokrotnie wpłynęła na zmianę koncepcji podczas wykonywania zadania oraz pomogła podjąć decyzje odnośnie technik stosowanych w celu osiągnięcia pożądanego rezultatu.

Pozytywnie należy również ocenić umożliwienie uczestnikom możliwości sprawdzania poprawności pisanych przez nich programów za pomocą zestawów testów jednostkowych, których wyniki mogły zasugerować zastosowanie alternatywnych rozwiązań oraz wskazać słabe punkty aktualnego kodu.

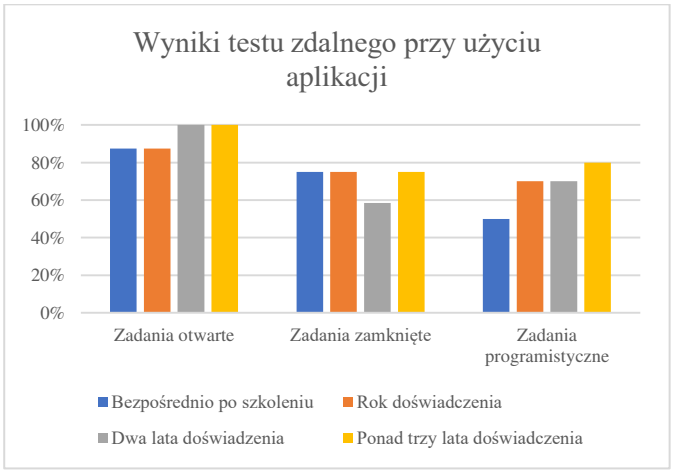
5.3. Zdalny test przeprowadzony przy użyciu stworzonej aplikacji

5.3.1. Założenia

Kolejną metodą, według której wykonano badania był zdalny test przeprowadzony przy użyciu przygotowanej aplikacji. W trakcie jego trwania uczestnicy mogli korzystać z dowolnych materiałów, ale zostali pouczeni o tym, że nie mogą się ze sobą komunikować. Modyfikacja wprowadzona w stosunku do poprzednio opisywanej metody miała na celu ukazanie rzeczywistego charakteru pracy programisty, który niejednokrotnie wspiera się różnymi źródłami, a także często ma dowolność, jeśli chodzi o czas i miejsce, w którym wykonuje swoją pracę.

5.3.2. Wyniki

Na Rys. 4 przedstawiono wyniki uzyskane ze zdalnie przeprowadzonego testu przy pomocy stworzonej aplikacji.



Rys. 4. Wyniki testu zdalnego przy użyciu aplikacji

5.3.3. Wnioski

Zadania teoretyczne w warunkach z dostępem do ogólno pojętych źródeł informacji zostały przez uczestników rozwiązywane bez większych problemów. Może to świadczyć o tym, że każdy z uczestników był w stanie zdobyć informacje niezbędne do udzielenia odpowiedzi na ww. pytania, co jest bardzo pożądaną cechą wśród programistów.

Dostrzeżono również fakt, iż uczestnicy badania, mając dostęp do dowolnych źródeł wiedzy, zdołali w krótszym czasie odpowiedzieć na pytania zamknięte i otwarte, dzięki czemu więcej czasu mogli poświęcić na dopracowanie rozwiązań zadań programistycznych.

Wyniki zadań programistycznych nie odbiegają znacznie od wyników przeprowadzonych z wykorzystaniem aplikacji w warunkach kontrolowanych. Można stwierdzić, że to przede wszystkim na ich podstawie należałoby oceniać badane osoby, ponieważ programowanie to umiejętność, którą nabywa się poprzez czasochłonną naukę oraz ćwiczenia. Nie jest możliwym znalezienie pełnego rozwiązania problemu programistycznego w ograniczonym czasie, gdy nie posiada się odpowiedniej wiedzy i umiejętności, a mając nawet nieograniczony dostęp do źródeł informacji.

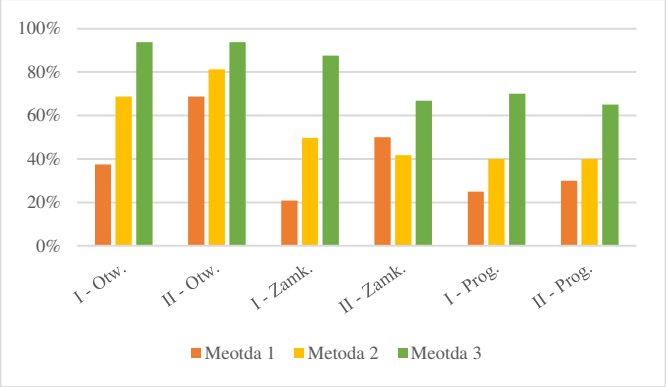
5.4. Porównanie wyników

Przeprowadzając i analizując badania wykonane różnymi metodami autorom pracy udało się zgromadzić dane, które mogą posłużyć do porównania poziomu skuteczności oceny

programistów, niezawodności działania, atrakcyjności użytkowania i uniwersalności stosowania.

5.4.1. Poziom skuteczności metod

Jako pierwszy czynnik, ocenie poddana została skuteczność weryfikacji wiedzy i umiejętności programistów. W tym celu uśrednione wyniki uzyskane przez uczestników testów w poszczególnych zadaniach zostały zestawione na wykresie przedstawionym na Rys. 2.



Rys. 5. Zestawienie średnich wyników w poszczególnych zadaniach w zależności od metody badawczej

Z Rys. 2 wynika, że istnieje dysproporcja pomiędzy wynikami zadań teoretycznych rozwiązywanych za pomocą różnych metod. Uczestnicy uzyskali najlepsze rezultaty w metodzie trzeciej - zdalnym teście przeprowadzonych w aplikacji. Przyczyn tego zjawiska można dopatrywać się w możliwości korzystania z materiałów pomocniczych w ww. metodzie. Dodatkowym czynnikiem może być również zredukowanie stresu poprzez umożliwienie zdawania testu w dogodnym miejscu i czasie. Nie można niestety również wykluczyć wpływu osób trzecich, które mogły pomagać rozwiązywać problemy badanym programistom.

Zauważalna jest także znaczna poprawa wyników zadań programistycznych przeprowadzonych przez aplikację względem testów pisemnych. Udogodnienia wynikające z możliwości kompilowania kodu, uruchamiania testów jednostkowych oraz podpowiadanie i kolorowanie składni na pewno pozytywnie wpłynęły na efektywność programistów. Podczas rozwiązywania testów pisemnych uczestnicy często się mylili, zmieniali koncepcje, próbowali różnych sposobów, ale zmagając się z aspektami technicznymi takimi jak nieznajomość interfejsów metod, niejednokrotnie zatrzymywali twórczy proces pisania kodu na rzecz prób rozwiązania trywialnych problemów ze składnią kodu.

Niepokojącą sytuacją natomiast może się wydawać fakt, iż wyniki uzyskane przez najmniej doświadczonych programistów w testach odbywających się za pośrednictwem aplikacji, nie odbiegały znacząco od rezultatów osiąganych przez osoby z najdłuższym stażem pracy. Sytuację tą można korygować odpowiednio dobierając poziom trudności zadań do poziomu zaawansowania kandydatów.

5.4.2. Niezawodność działania

Kolejnym kryterium oceny metod jest niezawodność ich działania. Niewątpliwie najlepiej w tej kategorii wypadają testy pisemne. Nie jest to metoda skomplikowana więc na ewentualne jej niepowodzenie wpływa najmniejsza liczba czynników. Odmienna sytuacja występuje w przypadku metod wykorzystujących aplikację przygotowaną na potrzeby pracy dyplomowej. Zachodzi w niej wiele procesów zależnych od siebie wzajemnie, zatem niepomyślne wykonanie jednego z nich może skutkować niepoprawnym działaniem aplikacji. Potencjalnych słabych punktów zaimplementowanego rozwiązania można dopatrywać się np. w komunikacji między modułami. Żądanie z aplikacji klienckiej musi dotrzeć do aplikacji serwerowej, ona zaś, musi się połączyć z bazą danych i pobrać kod z repozytorium Git udostępnionego na platformie GitLab, dysponując niezbędnymi danymi, aplikacja serwerowa wysyła odpowiedź do aplikacji klienckiej. Na każdym etapie ww. procesu może wystąpić zdarzenie, które spowoduje, że komunikacja zostanie zerwana. Pomimo tego, podczas testów nie odnotowano przypadków niepoprawnego działania aplikacji. Rozwiązaniem na potencjalne problemy wynikające z zakłóceń komunikacji pomiędzy modułami może być własna implementacja wszystkich modułów i tym samym uniezależnienie się od zagrożenia awaryjnością usług zewnętrznych dostawców.

5.4.3. Atrakcyjność użytkowania

Jako następny czynnik wpływający na ocenę metody, została oceniona atrakcyjność użytkowania aplikacji. Podczas badań od uczestników udało się zebrać cenne opinie na temat działania aplikacji, a także sugestie dotyczące jej usprawnień.

Na podstawie analizy zachowań osób biorących udział w eksperymencie oraz bezpośrednio otrzymywanych od nich uwag udało się ustalić, że nie były entuzjastycznie nastawione do testu pisemnego. Większość sądziła, że pisanie programów na kartce papieru jest niewygodne, bardziej czasochłonne i przyczynia się do popełniania błędów. Kolejnym argumentem był czas, który mieli poświęcić na rozwiązanie testu. Programiści niechętnie poświęcili godzinę swojego prywatnego czasu na udział w badaniach, ale mimo to przekonali się, że trudno było rozwiązać wszystkie zadania na czas.

Odmienne zdanie mieli natomiast uczestnicy badań z udziałem aplikacji, ponieważ nie kojarzyli testu z przykrym obowiązkiem, a traktowali go jak wyzwanie. Mogło to wynikać z ciekawości jak działa aplikacja, czy chęcią sprawdzenia nowego sposobu przeprowadzania testów. Po uczestnikach dało się zaobserwować większe rozluźnienie podczas rozwiązywania zadań, a poziom zaimplementowanych programów zdawał się odzwierciedlać nastrój programistów, gdyż były bardziej uporządkowane i przemyślane, niż w przypadku testów pisemnych.

Testy zdalne nie były obserwowane, ale z relacji uczestników wynika, że nie doświadczyli problemów

z działaniem aplikacji, a ich testy przebiegły bez zakłóceń. Od tych osób również otrzymano bardzo pozytywne opinie. Najbardziej chwalona była możliwość uruchamiania testów w trakcie rozwiązywania zadania, ponieważ wykazywały one pewne trudne do przewidzenia przypadki oraz niedociągnięcia programów. Na podstawie wyników testów programiści z łatwością identyfikowali i eliminowali wszelkie błędy. Uczestnicy lepiej kontrolowali czas rozwiązywania testu dzięki stale widocznemu zegarowi, a aplikacja uniemożliwiała jego przekroczenie, co miało miejsce w przypadku testów pisemnych.

6. Podsumowanie

Wybrany temat badań okazał się być niezwykle ciekawym zagadnieniem zarówno pod względem teoretycznym, jak i praktycznym. Autorom udało się przede wszystkim zgłębić wiedzę na temat dostępnych na rynku metod i zasadach ich działania, a także o ich docelowemu przeznaczeniu, wykorzystywanych przez nie technologiach oraz mocnych i słabych stronach każdej z nich.

Niezwykle interesującym wyzwaniem pod kątem technicznym była próba stworzenia prototypu aplikacji implementującej m.in. logikę automatycznej weryfikacji poprawności rozwiązania zadania programistycznego, którą można uznać za zakończoną wielkim sukcesem. Badania przeprowadzone z wykorzystaniem przygotowanego prototypu i przy udziale ochotników pracujących na co dzień w zawodzie programisty pozwoliły spojrzeć na proces weryfikacji wiedzy programistycznej z całkiem innej perspektywy. Dostrzeżono składowe całego procesu oraz analizowanych metod podatne na błędy ludzkie i niedoskonałości zaimplementowanych rozwiązań, które mogą skutkować znacznymi opóźnieniami, dodatkowymi kosztami, a przede wszystkim błędną oceną wiedzy i umiejętności badanej osoby.

Szacuje się również, że metody weryfikacji wiedzy i umiejętności programistów w najbliższym czasie wciąż będą rozwijane. Firmy oferujące tego typu rozwiązania będą dążyły do osiągnięcia jak najwyższej niezawodności, wszechstronności oraz zmniejszenia poziomu ingerencji osoby nadzorującej aplikację do niezbędnego minimum. Z tego typu systemów coraz częściej będą korzystały uczelnie szkolące przyszłych programistów oraz firmy poszukujące już wykwalifikowanych specjalistów.

Literatura

[1] Sedlak & Sedlak, „Prognozy wzrostu zatrudnienia i popytu w branży IT i telekomunikacji - Rynek Pracy” (2014), https://rynekpracy.pl/monitory/prognozy-wzrostu-zatrudnienia-i-popytu-w-branzy-it-i-telekomunikacji. [23.06.2018]

[2] S. Shahida, R. Rohaida i Z. Z. Kamal, „Improving Automated Programming Assessments: User Experience Evaluation Using FaSt-generator,” w The Third Information Systems International Conference (2015).

[3] K. Cooper i L. Torczon, Engineering a Compiler 2nd Edition, Elsevier, 2011

[4] A. V. Aho, R. Sethi i J. D. Ullman, Kompilatory. Reguły, metody i narzędzia, WNT, 2002

[5] R. Osherove, The art of Unit Testing, Manning Publications, 2013

[6] T. Kaczanowski, Złe testy, dobre testy, 2016.

[7] M. Janicki i K. Strzecha, Zastosowanie statycznej analizy do walidacji kodu języka Java, Wydawnictwa AGH, 2004.

[8] G. A. Campbell, "Cognitive Complexity. A new way of measuring undertandability," SonarSource, (2018), https://www.sonarsource.com/docs/CognitiveComplexity.pdf.

[9] Codility, „The Codility Task Library | Codility Help Center” (2018), http://support.codility.com/screening-candidates-with-codecheck/the-codility-task-library [23.06.2018]

[10] M. Jackson, S. Crouch i R. Baxter "Software Evaluation: Criteria-based Assessment" Software Sustainability Institute, 2011