KORZYŚCI WYNIKAJĄCE Z ZASTOSOWANIA STRATEGII EWOLUCYJNYCH W PROCESIE GENEROWANIA DANYCH TESTOWYCH

Marek ŻUKOWICZ
Politechnika Rzeszowska

Streszczenie: Celem artykułu jest podkreślenie zalet, które można uzyskać poprzez zastosowanie strategii ewolucyjnych w testowaniu, a konkretnie w procesie generowania danych testowych. Pierwszy rozdział wprowadza czytelnika do tematu artykułu. Prezentuje informacje na temat problemu jakości, przydatności danych testowych oraz kryteria jakości danych testowych. Drugi rozdział stanowi przegląd publikacji, w których poruszony jest problem generowania danych testowych za pomocą strategii ewolucyjnych. Zaprezentowane są w tym rozdziale różne podejścia rozwiązujące problem optymalizacji danych testowych. Trzeci rozdział przedstawia korzyści, które zdaniem autora wynikają z zastosowania strategii ewolucyjnych w procesie generowania danych testowych. Wyciągnięte również zostały wnioski z artykułu, książek wymienionych w bibliografii oraz autora artykułu, jako osoby testującej praktycznie. Ostatni rozdział oprócz podsumowania i wniosków, zawiera propozycje autora sugerujące dla jakich problemów związanych z testowaniem warto zastosować strategie ewolucyjne.

Słowa kluczowe: strategie ewolucyjne, dane testowe, kryteria jakości danych testowych, optymalizacja danych testowych.

WPROWADZENIE

Dane testowe to wartości, które wprowadza się do testowanego systemu jako dane wejściowe dla pewnej funkcjonalności w celu weryfikacji lub walidacji jej poprawności. Dane wejściowe są nierozerwalnie związane z przypadkiem testowym [18]. Są to informacje o tym, jakie wartości należy podstawić do funkcji realizującej wymagania w systemie. Przypadek testowy to pojedyncza czynność polegająca na sprawdzeniu pewnych właściwości oprogramowania lub urządzenia.

W literaturze istnieje podział danych testowych na dane wymagane oraz dane niewymagane. Autorzy pozycji [18] piszą, że dane testowe wymagane to takie, których podanie jest niezbędne w procesie użytkowania aplikacji. W praktyce łatwo zauważyć, że aplikacje komputerowe odmiennie reagują na błędnie wprowadzane dane lub po prostu zawierają różne rodzaje błędów, które uniemożliwiają prawidłowe funkcjonowanie aplikacji tuż po wprowadzeniu danych do formularza, bez względu na to, czy te dane są wymagane czy nie. Dla każdej aplikacji istnieją takie zestawy danych, które z dużym prawdopodobieństwem zostaną wprowadzone podczas użytkowania aplikacji oraz takie, których użytkownik prawdopodobnie nie wprowadzi przez cały cykl życia oprogramowania [17]. Niezależnie od tego, jakie dane testowe wprowadzane są jako dane wejściowe, aplikacja musi zareagować na nie pozytywnie, a to oznacza, że dane zostaną zapisane, przetworzone, przeniesione lub aplikacja zwróci komunikat o niepoprawnych danych, który dodatkowo poinformuje użytkownika, które pole zawiera niedozwolone dane lub jaka funkcja dostała niepoprawne dane wejściowe. Istnieje wiele technik, które pozwalają ułatwić wybór danych tak, aby były one zgodne z danymi rzeczywistymi, czyli takimi, które będę wprowadzane do aplikacji przez

potencjalnych użytkowników. W pozycji [19] podane są techniki pozyskiwania danych testowych jak:

- Testowanie dziedziny,
- Podział na klasy równoważności,
- Analiza wartości brzegowych (można analizować wartości brzegowe wejściowe oraz wyjściowe),
- Testowanie porycia kodu (testowanie pokrycia instrukcji, testowanie okrycia warunków logicznych),
- Zastosowanie tablic decyzyjnych,

Inne metody pozyskiwania danych testowych to m. in.:

- Zastosowanie metody pair-wise testing [22],
- Zastosowanie metody n-wise testing,
- Wykorzystanie ogólnych lub własnych matematycznych modeli funkcjonalności w systemach informatycznych [21, 23, 24],

Istnieje w testowaniu oprogramowania takie pojęcie jak strategia testowa, czyli pewien ciąg czynności pozyskiwania informacji o przypadkach testowych czyli również na temat danych testowych. W literaturze wyróżnia się formalne strategie testowe (zgodne z międzynarodowymi normami jakości) oraz nieformalne strategie testowe (dobieranie danych w oparciu o doświadczenie testerów, zgadywanie błędów związanych z wprowadzanymi danymi). Strategie testowe mogą być mieszane z technikami pozyskiwania danych wejściowych. Wg Farley`a, Humble`a oraz Romana [6, 20] dane testowe powinny w szczególności odznaczać się takimi cechami jak:

- Wydajność (może być rozumiana jako czas wykonania testu),
- Zależność od systemu (jakość zależy od aspektów technologicznych),
- Zupełność (stopień, w jakim te dane mają wartość dla wszystkich atrybutów w systemie),
- Aktualność,
- Wiarygodność,
- Spójność (np. niesprzeczność pomiędzy danymi testowymi),
- Precyzyjność (dotyczy testów na liczbach, w aplikacjach które wymagają dużej dokładności np. przeliczanie miar czy jednostek).

Poprawne dane testowe można również uzyskać z Internetu lub poprzez zastosowanie generatorów danych, które są darmowymi narzędziami.

W ostatnich latach wraz z rozwojem strategii ewolucyjnych pojawiły się pomysły ich zastosowania w procesie generowania danych testowych. Początki takich prac sięgają drugiej połowy lat 90 np. pozycja [10]. Strategie ewolucyjne dzielą się na programowanie ewolucyjne, algorytmy genetyczne oraz algorytmy ewolucyjne. Różnice między algorytmami genetycznymi oraz algorytmami ewolucyjnymi nie są duże. Bardzo często te nazwy są różnie stosowane w literaturze niezależnie od konstrukcji algorytmu. Początkowo algorytmy genetyczne bazowały na ciągach binarnych, natomiast algorytmy ewolucyjne na zapisie zmiennoprzecinkowym [19]. Celem napisania artykułu jest przegląd zastosowania strategii ewolucyjnych w procesie generowania danych testowych oraz wskazanie wartości jakie mogą przynieść strategie ewolucyjne w procesie testowania aplikacji. Autor zakłada, że czytelnik

zna podstawowe pojęcia takie jak graf, algorytm, optymalizacja, ekstremum funkcji, strategia ewolucyjna oraz zna podstawy informatyki.

PRZEGLĄD WYBRANYCH STRATEGII EWOLUCYJNYCH W PROCESIE GENEROWANIA DANYCH TESTOWYCH

Jak już wspomniano w poprzednim rozdziale, początki prób zastosowania strategii ewolucyjnych sięgają drugiej połowy lat 90-tych. Pozycja [10] przedstawia strategię, w której za pomocą kodu pewnego programu generowane są grafy przepływu danych CFG (control flow graph) oraz grafy zależności danych CDG (control dependence graph). Algorytm genetyczny działa w taki sposób, że generuje pewien zestaw przypadków testowych i sprawdza, czy wygenerowane przypadki spełniają warunki stopu. Jeśli nie są spełnione warunki stopu, to algorytm losuje nowy zestaw osobników, z którego będą generowane kolejne osobniki, czyli dane testowe. Ograniczenie dla algorytmu stanowi również czas. Mierzenie jakości odbywa się poprzez pokrycie ścieżek w grafie CDG poprzez wygenerowane dane testowe. Strategia została porównana z algorytmem, który generuje dane testowe losowo. Okazało się, że algorytm genetyczny w krótszym czasie potrafi rozwiązać problem pokrycia ścieżek grafu CDG niż algorytm losowy. Analogiczne podejście jest zaprezentowane w pozycji [14].

W pozycjach [1, 5, 7, 13] autorzy przedstawili strategie ewolucyjne, które generują dane testowe poprzez wykorzystanie grafów. Autor pozycji [1] proponuje zastosowanie algorytmu genetycznego, który generuje ciągi ścieżek testowych opartych na grafie przepływu danych. Wprowadzone zostały w artykule definicje niezależnej ścieżki, podstawowej ścieżki oraz podstawowego zbioru ścieżek. Ścieżka niezależna to taka, w której każda krawędź nie występuje w innych ścieżkach w grafie przepływu danych. Podstawowy zbiór ścieżek to taki, który spełnia trzy warunki: każda ścieżka jest niezależna, wszystkie krawędzie w grafie powinny być pokryte przez podstawowy zbiór, każda ścieżka nie zawarta w podstawowym zbiorze może być złożona z kombinacji liniowej ścieżek z podstawowego zbioru. Ciekawą cechą algorytmu jest zmienna długość elementów, które powstają w wyniku zastosowania operatora krzyżowania. W pozycji [5] opisany został algorytm, który działa w ten sposób, że jego celem nie jest znalezienie jednego rozwiązania, natomiast kilku rozwiązań jednocześnie. Wspomniane rozwiązanie to zbiór węzłów w grafie przepływu danych, które są zdefiniowane jako cele, które powinien osiągnąć algorytm. Dostarczona populacja wejściowa (dane wejściowe) jest dzielona na podzbiory, z których wybierany jest najlepszy względem dopasowania element. Zbiór najlepszych elementów to z kolei populacja, dla której algorytm sprawdza osiągalność założonych celów. Dane testowe (wygenerowane elementy) za pomocą których można dojść do założonego celu są dodawane to zbioru przypadków testowych. W wyniku kolejnych iteracji algorytm dodaje do zbioru danych testowych takie dane wejściowe, które wyznaczają nowy nieosiągnięty dotąd cel przez poprzednio wygenerowaną populację. Autorzy pozycji [7] opisują strategię generowania danych testowych analogiczną do strategii opisanej w pozycji [5]. Różnice pojawiają się natomiast w zastosowaniu funkcji dopasowania, oraz w sposobie wybierania osobników z podzbiorów, które powstały z dostarczonych początkowych danych wejściowych. Algorytm ewolucyjny opisany w pozycji [7] działa równolegle przeszukując każdy podzbiór, przez co wzrasta szybkość znalezienia rozwiązania. W pozycji [13] zaprezentowano strategię, w której następuje łączenie diagramu UML z

diagramem SD (Sequence diagram), czyli ciągiem wykonywanych instrukcji umożliwiających dojście do założonego celu (w tym przypadku węzła w grafie). Problem dotyczy wskazania takich ścieżek (czynności, które wykonuje się podczas testów), które powinny być przetestowane w pierwszej kolejności. Artykuł przedstawia trzy algorytmy: pierwszy z nich proponuje generowanie danych wejściowych z diagramu nazwanego diagram czynności (activity diagram), drugi proponuje przyporządkowanie wag do węzłów, ostatni zaś zamienia diagram czynności na diagram stanów (state diagram) w celu umożliwienia procesu priorytetyzacji ścieżek testowych (tutaj danych testowych). Funkcja dopasowania zastosowana w algorytmie genetycznym wykorzystuje w procesie ewolucji wagi przypisane do ścieżek w grafie. Ścieżka jest zamieniana na ciąg binarny, z którego algorytm ewolucyjny oblicza wartość funkcji dopasowania, stosuje na tym ciągu krzyżowanie oraz mutację. Jeśli algorytm znajdzie taki ciąg, który osiągnie najlepszą wartość dopasowania dla danych wejściowych, to wtedy ten ciąg jest wskazany jako najbardziej optymalny. Optymalizacją w tym przypadku jest minimalizacja kwadratu obliczonego kosztu ścieżki. W pozycji [2] autorzy przedstawiają strategię, która łączy algorytm grupowania danych K-means z algorytmem genetycznym. Dostarczone dane wejściowe są zamieniane na liczby, w celu umożliwienia zastosowania na nich metody grupowania K-means. Po podzieleniu danych na podzbiory przez algorytm K-means, zaczyna działać algorytm genetyczny. Każda klasa posiada środek ciężkości. Zadaniem algorytmu genetycznego jest wygenerowanie takiego podzbioru danych wejściowych, żeby nie trafiły do niego dwa punkty o zbliżonych współrzędnych. Autorzy pozycji [3] przedstawili propozycję implementacji algorytmów opartych na strategiach ewolucyjnych, w których proces optymalizacji generowania danych testowych rozpoczyna się od dostarczenia pewnej populacji początkowej (danych testowych), po czym następuje mierzenie pokrycia instrukcji w kodzie za pomocą funkcji dopasowania. W oparciu o wartość funkcji dopasowania algorytm wybiera elementy populacji, które biorą udział w reprodukcji. Autorzy porównali zastosowanie algorytmu genetycznego, algorytmu ewolucyjnego oraz drugiego algorytmu ewolucyjnego zaimplementowanego równolegle. Główną różnicą pomiędzy omówionymi w artykule algorytmami jest proces mutacji oraz krzyżowania. Pozycja [4] jest o tyle ciekawa, że stanowi połączenie algorytmu stosowanego w filtrze Kalmana znanego z automatyki, z algorytmem ewolucyjnym, który rozwiązuje problem optymalizacji generowania danych testowych. Skrócona nazwa opisanego w artykule algorytmu to KFGA. Autorzy przyjmują założenie, że wygenerowane dane wejściowe powinny pokryć jak najwięcej linii kodu poprzez wywołanie odpowiednich instrukcji w testowanym oprogramowaniu, ale przy takim ograniczeniu, żeby algorytm wygenerował jak najmniej przypadków testowych (danych wejściowych). Algorytm KFGA jest algorytmem adaptacyjnym, dzięki czemu wyniki działania tego algorytmu są na ogół zadowalające, jednak nie zawsze są najlepsze, co widać w tabelach, w których autorzy porównali inne znane wcześniej strategie ewolucyjne z algorytmem KFGA. Artykuły wymienione w pozycjach [8] oraz [9] przedstawiają bardzo podobne strategie, które polegają na poszukiwaniu błędów w kodzie testowanego programu. Różnią się one głównie tym, że w pozycji [9] algorytm ewolucyjny przeszukuje kilka ścieżek jednocześnie. Błędy podzielone są na kategorie, znajdywanie błędów jest oceniane pod kątem tych kategorii. W pozycji [11] autorzy prezentują ciekawe podejście zastosowania algorytmu ewolucyjnego, w którym warunki stopu są dynamiczne, w tym przypadku uzależnione od jakości wygenerowanych danych

testowych. Algorytm sam decyduje, kiedy zakończyć proces generowania danych testowych. Zastosowana jest tutaj strategia modelu spadku ryzyka awarii. Pozycja [12] to głównie przegląd metod SBST (Self Based Testing Model), które generują dane testowe wg pewnych trzech różnych strategii opartych na genetycznych algorytmach. Artykuł powstał w wyniku grantu naukowego. Zawarte w nim treści dotyczą głównie testów funkcjonalnych związanych z testami pokrycia kodu. Autorzy pozycji [15] pokazują, jak można wykorzystać algorytm ewolucyjny w celu priorytetyzacji danych testowych (tutaj przypadków testowych), biorąc pod uwagę czas wykonania testu oraz ilość defektów, które znajduje dany przypadek testowy. Autor pozycji [16] prezentuje algorytm ewolucyjny, który wykorzystuje strategię o nazwie Harmony Search w procesie optymalizacji generowania danych testowych do testów strukturalnych. Działanie algorytmu ewolucyjnego rozpoczyna się od procesu inicjalizacji pamięci harmonicznej (harmony memory) czyli kroku, w którym losowo generowane są wektory danych testowych (wektory n-wymiarowe). Nowy wektor danych generowany jest wg trzech reguł: wyciąganie danych z pamięci algorytmu, zupełnie losowo oraz w sposób kontrolowany przez twórcę. Pozostałe kroki strategii ewolucyjnej opisanej w artykule są analogiczne do tradycyjnych algorytmów ewolucyjnych. Okazuje się, że taka strategia w niektórych przypadkach testowania strukturalnego daje szybkie i dobre rozwiązanie.

ZALETY STOSOWANIA STRATEGII EWOLUCYJNYCH W GENEROWANIU DANYCH TESTOWYCH

Analiza poprzednich dwóch rozdziałów pozwala wyciągnąć pewne wnioski, które wynikają z zastosowania strategii ewolucyjnych w procesie generowania danych testowych:

- Losowość ale taka, której kierunek jest kontrolowany za pomocą funkcji dopasowania oraz procesów ewolucyjnych (krzyżowanie i mutacja). Istnieją generatory danych testowych, ale generują dane wejściowe w sposób całkowicie niekontrolowany. Z większości pozycji publikacji zawartych w bibliografii wynika, że zastosowanie operatorów ewolucyjnych oraz funkcji dopasowania pozwala wygenerować dane testowe w taki sposób, że są one zgodne z przeznaczeniem, przez co również są wiarygodne, zależne od testowanego systemu oraz niesprzeczne,
- Wprowadzanie mutacji, którą można potraktować jako czynnik dający możliwość wyeliminowania zjawiska powtarzania pewnego schematu podczas generowania danych testowych, a z matematycznego punktu widzenia mutacja może uchronić od zjawiska utknięcia w obszarze ekstremum lokalnego,
- Metoda zastosowania algorytmu ewolucyjnego jest dość uniwersalna, wobec czego może być wykorzystana do generowania danych testowych w oparciu o różne modele funkcjonalności w systemach informatycznych. Dla każdego modelu matematycznego lub diagramu przypadków użycia można zastosować algorytm ewolucyjny w procesie generowania danych testowych,
- Metoda jest stosunkowo szybka: znalezienie rozwiązania jest często możliwe po kilku iteracjach lub przeszukaniu niewielkiej części stanów, co widać w publikacjach zawartych w bibliografii [2, 3, 4],
- Odporność na paradoks pestycydów. Paradoks pestycydów w testowaniu mówi, że te same testy powtarzane bez żadnych zmian, stają się coraz mniej skuteczne w znajdowaniu

usterek w testowanym programie [20]. Dzięki zastosowaniu strategii ewolucyjnej do wygenerowania danych wejściowych ryzyko takiego zjawiska znacznie zmaleje lub w ogóle nie wystąpi,

- Możliwość zdefiniowania funkcji dopasowania w taki sposób, aby wygenerowane dane testowe były bardzo związane z kontekstem użycia testowanego oprogramowania. Testowanie jest zależne od kontekstu, więc dane testowe również muszą być zależne od kontekstu użycia aplikacji,
- Optymalizacja wielokryterialna. Funkcja dopasowania, która wyznacza kierunek generowania nowego pokolenia (tutaj danych testowych), może być sumą kilku funkcji mierzących jakość, które mogą być przemnożone przez wagi. Daje to możliwość wygenerowania danych, które są poprawne dla kilku atrybutów związanych z danymi,
- Możliwość wprowadzania adaptacji, czyli zmiennych kryteriów stopu algorytmu. W przypadku, gdy algorytm nie znajdzie zadowalającego rozwiązania względem dopasowania do postawionego problemu i określonej funkcji celu, możliwa jest zmiana parametrów, które kontrolują proces generowania nowego pokolenia (w tym przypadku danych testowych), co może poprawić jakość rozwiązania postawionego problemu [20],
- Możliwość łączenia z innymi algorytmami (K-means, Filtr Kalmana), co zwiększa łatwość rozwiązania problemu optymalizacji [4, 7].

PODSUMOWANIE

Głównym założeniem napisania artykułu było zebranie informacji na temat znanych powszechnie technik generowania danych testowych, przedstawienie sposobów generowania danych testowych za pomocą strategii ewolucyjnych (mało znanych przez osoby testujące w praktyce) oraz przedstawienie płynących z tego podejścia korzyści. Podczas głębszej analizy artykułu, czytelnik z pewnością zda sobie sprawę z tego, że testowanie jak i dobieranie danych testowych nie jest trywialną czynnością. Publikacje na ogół zawierają proste przykłady zastosowania oraz małe odniesienie do praktyki, ponieważ nie są opisywane badania naukowe na systemach używanych np. przez firmy prywatne lub instytucje publiczne. Opisane w drugim rozdziale publikacje prezentują jednak strategie rozwiązywania bardzo ważnego problemu w testowaniu. Analizując artykuły zawarte w bibliografii, czytelnik z pewnością zauważy, strategie ewolucyjne dobrze sobie radzą z problemem optymalizacji generowania danych testowych. Warto wobec tego podjąć próby stosowania takich strategii w praktyce, przeprowadzając testy na używanych lub rozwijających się systemach informatycznych. W przyszłości pojawią się ze strony autora artykułu propozycje zastosowania strategii ewolucyjnych w testowania systemów informatycznych, które są używane przez przedsiębiorstwa oraz ewoluują w sposób ciągły, dostosowując się do zmieniającego się środowiska.

„Poprzeczka jakości" systemów informatycznych wzrasta z biegiem lat. Algorytm ewolucyjny poprzez zastosowanie krzyżowania, mutacji oraz funkcji dopasowania może tak wygenerować dane testowe, aby odznaczały się takimi cechami jakości, które są bardzo ważne w testowaniu.

LITERATURA

1. S.G. Ahmed. *Automatic generation of basis test paths using variable length genetic algorithm,* Journal Information Processing Letters, Volume 114 Issue 6, June, 2014, pp. 304-316.
2. R. Alavi, S. Lofti. *The New Approach for Software Testing Using a Genetic Algorithm Based on Clustering Initial Test Instances*, International Conference on Computer and Software Modeling 2011, IPCSIT vol.14 (2011).
3. E. Alba, F. Chicano. *Observations in using parallel and sequential evolutionary algorithms for automatic software testing*, Computers & Operations Research, Vol. 35 Issue 10, October 2008, pp. 3163-3183.
4. A. Aleti, L. Grunske. *Test data generation with a Kalman filter-based adaptive genetic algorithm,* The Journal of Systems and Software Vol. 103, May 2015 pp. 343-352.
5. M. Alshraideh, B.A. Mahafzah, S. Al-Sharaeh. *A multiple-population genetic algorithm for branch coverage test data generation*, Software Quality Journal, Vol. 19, Volume 19, Issue 3 September 2011, pp. 489-513.
6. D. Farley, J. Humble. *Ciągłe dostarczanie oprogramowania*, Helion 2015.
7. D. Gong, T. Tian, X. Yao. *Grouping target paths for evolutionary generation of test data in paralel*, The Journal of Systems and Software, Vol. 85, Issue 11, November 2012, pp. 2531–2540.
8. D. Gong, Y. Zhang. *Generating test data for both path coverage and fault detection using genetic algorithms* , Frontiers of Computer Science, December 2013, Vol. 7, Issue 6, pp. 822-837.
9. D. Gong, Y. Zhang. *Generating test data for both path coverage and fault detection using genetic algorithms: multi-path case*, Frontiers of Computer Science, October 2014, Vol. 8, Issue 5, pp. 726-740.
10. M.J. Harrold, R. Pargas, R. R. Peck. *Test-Data generation using genetics algorithms,* Journal of Software Testing, Verification and Reliability 1999.
11. I. Hermadi, C. Lokan, R. Sarker. *Dynamic stopping criteria for search-based test data generation for path testing* , Information and Software Technology, April 2014 Vol. 56, pp. 395-407.
12. J. Hudec, E. Gramatova. *An Efficient functional test generation method for processors using genetic algorithms*, Journal of Electrical Engineering, July 2015, Vol. 66, Issue. 4, pp. 186-193.
13. N. Khurana, R.S. Chillar. *Test Case Generation and Optimization using UML Models and Genetic Algorithm*, Procedia Computer Science, August 2015, Vol. 57, pp. 966-1004.
14. H. Kim, P.R. Srivastava. *Application of Genetic Algorithm in Software Testing*, International Journal of Software Engineering and Its Applications, October 2009, Vol. 3, Issue 4, pp. 87-96.
15. R. Krishnamoorthi, A. Sahaaya, S.A. Mary. *Regression Test Suite Prioritization using Genetic Algorithms,* International Journal of Hybrid Information Technology, July 2009 Vol. 2, Issue 3, July.

16. C. Mao. *Harmony search-based test data generation for branch coverage in software structural testing*, Neural Computing and Applications, September 2013, Vol. 25, Issue 1, pp.199-216. Springer-Verlag London 2013.
17. M. Mirzaaghaei, F. Pastore, M. Pezze. *Automatic test case evolution,* Software testing, Verification and Reliability, April 2014, Vol. 24, Issue 5, pp.386-411.
18. A. Piaskowy, R. Smilgin. *Dane Testowe, teoria i praktyka*, Helion 2011.
19. A. Roman, *Testowanie i jakość oprogramowania*, PWN 2015.
20. D. Rutkowska, M. Piliński, L. Rutkowski. Sieci neuronowe, algorytmy genetyczne i systemy rozmyte, PWN W-wa 1997.
21. D. Warchoł, M. Żukowicz. *Testing education: test case prioritization using matrices,* General and Professional Education, May 2015, Vol. 1, Issue 8, pp. 57-62.
22. M. Żukowicz. *Edukacja testowania: Narzędzie All-pairs Testing w procesie optymalizacji testów konfiguracji – zastosowanie narzędzia w systemie B2B OPTIbud,* General and Professional Education, Dezember 2015, Vol. 4, Issue 12, pp. 99-106.
23. M. Żukowicz. *Edukacyjne i ekonomiczne aspekty zastosowania cyklu Hamiltona w projektowaniu i testowaniu oprogramowania*, General and Professional Education, numer Dezember 2014, Vol. 4, Issue 13, pp. 95-102.
24. M. Żukowicz, O pewnych problemach analizy wartości brzegowych, Internet:, http://testerzy.pl/materialy/index.php?file=analiza-wartosci-brzegowych.pdf, data dostępu 03.03.2016

Data przesłania artykułu do Redakcji: 02.2016
Data akceptacji artykułu przez Redakcję: 04.2016

Mgr Marek Żukowicz
Politechnika Rzeszowska, Wydział Elektrotechniki i Informatyki
Katedra Automatyki i Informatyki
ul. Wincentego Pola 2, 35-959 Rzeszów, Polska
e-mail: bobmarek@o2.pl