JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 8 (2018) 252-257
Wysłane: 2018-05-27
Przyjęte: 2018-06-12

Porównanie narzędzi automatyzacji testów z wykorzystaniem interfejsu użytkownika na przykładzie Sikuli i AutoIT

Tomasz Paczuski*, Beata Pańczyk
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Celem artykułu jest porównanie narzędzi do automatyzacji testów interfejsu użytkownika na przykładzie Sikuli oraz AutoIT. W przeprowadzonych badaniach skoncentrowano się na czasie wykonywania skryptów testowych, ich złożoności, łatwości utrzymania oraz niezawodności. Na potrzeby badań stworzono aplikację w języku C# oraz napisano kilka reprezentatywnych testów automatycznych w każdym z narzędzi.

Słowa kluczowe: Sikuli; AutoIT; testy automatyczne

*Autor do korespondencji.

Adres e-mail: tpaczuski04@gmail.com

Comparison of tools for automated tests of the graphical user interface using the the Sikuli and AutoIT example

Tomasz Paczuski*, Beata Pańczyk
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The aim of the article is to compare the tools for automating user interface tests on the example of Sikula and AutoIT. The conducted research has focused on the time of testing scripts, their complexity, ease of maintenance and reliability. For the purposes of the study, an application in C # was created and several representative automatic tests were written in each tool.

Keywords: Sikuli; AutoIT; automated testing

*Corresponding author.

E-mail address: tpaczuski04@gmail.com

1. Wstęp

W dzisiejszym świecie oprogramowanie towarzyszy nam na każdym kroku. Można je spotkać praktycznie w większości urządzeń codziennego użytku. Niepoprawnie działające oprogramowanie może doprowadzić do utarty zaufania do producenta systemu, a co za tym idzie strat finansowych. Dlatego też firmy wytwarzające oprogramowanie zaczęły przywiązywać większą wagę do jakości wytwarzanego systemu. W sylabusie ISTQB [1] podane są intencje testów. Norma IEEE 829 [2] opisuje testowanie jako proces oparty o dokumentację oraz zawiera opis elementów planu testu. Według Myers [3] testowanie jest procesem uruchamiania oprogramowania w celu znajdywania błędów. Jednak nie do końca tak jest. Testowanie nie powinno tylko skupiać się na samym oprogramowaniu, ale również dodatkowo na innych elementach procesu wytwórczego [4].

Aby usprawnić proces testowy można wdrożyć narzędzia. Jak podaje [5] narzędzia do automatyzacji można wykorzystać np. do wykonania testów poprzez uruchomienie skryptów testowych, porównania oczekiwanych wyników testów z rzeczywistymi. Wdrożenie narzędzia do organizacji jest trudnym procesem, niesie ze sobą szereg korzyści, jak i ryzyk. Jak podaje [1], [6] przykładowymi ryzykami mogą być np. błędne oszacowanie kosztów oraz czasu wdrożenia narzędzia, błędne oczekiwanie w stosunku co do użycia narzędzia. Natomiast przykładowymi korzyściami wynikającymi ze wsparcia procesu testowego przez narzędzia to, np. wykonywanie testów zawsze w tej samej kolejności, redukcja monotonności pracy [6]. Wdrażając wybrane narzędzie - firma musi liczyć się również z kosztami. Przykładowe koszty, jak podaje [5], [7], to koszt opłat licencji i/lub wsparcia technicznego, koszt zakupu narzędzia, koszt związany z wytworzeniem własnego narzędzia.

Automatyzacja testów ma swoje metryki, dzięki którym można ocenić zyski lub przewidzieć kiedy testy zaczną je przynosić. Sylabus ISTQB dla inżyniera automatyzacji [8] przytacza m.in. takie metryki jak: czas trwania testów automatycznych, pracochłonność utrzymania testów, liczbę testów z wynikiem pozytywnym oraz negatywnym.

2. Przedmiot badań

Przedmiotem badań były dwa narzędzia do automatyzacji testów z wykorzystaniem interfejsu użytkownika. Wybrane narzędzia to SikuliX oraz AutoIT v3. Pierwsze z nich jest darmowe i działa pod systemami operacyjnymi Windows, Mac, Linux/Unix. Wykorzystuje rozpoznawanie obrazu oparte na bibliotece OpenCV do identyfikacji i sterowania

komponentami GUI. Narzędzie wspiera takie języki jak: Ruby, JavaScript, Python [9], [10].

AutoIT jest również darmowym narzędziem. Oparty jest o język podobny do BASIC. Przeznaczony jest jednak tylko na system Windows. Automatyzacja w tym narzędziu polega na symulacji użycia myszki oraz klawiatury. Napisane skrypty można kompilować do plików exe [11], [12].

3. Przeprowadzone badania

W celu porównania narzędzi została stworzona aplikacja desktopowa w języku C#. Przedmiotem badań są testy GUI. Aplikacja podzielona jest na moduły: użytkownika, pacjenta, lekarza. Aplikacja umożliwia:

- zalogowanie się,
- dodanie użytkownika,
- dodanie pacjenta,
- dodanie lekarza,
- rejestrację pacjenta.

Napisane skrypty testują poprawność działania interfejsu użytkownika w zakresie:

- test 1 - logowania na konta użytkownika admin,
- test 2 - niepoprawnego logowania,
- test 3 - wylogowania,
- test 4 - dodania użytkownika,
- test 5 - niepoprawnego dodania użytkownika,
- test 6 - dodania lekarza,
- test 7 - dodania pacjenta,
- test 8 - rejestracji pacjenta,
- test 9 - niepoprawnej rejestracji pacjenta.

Każdy z powyższych testów został uruchomiony 20 razy.

3.1. Kryteria porównania narzędzi

Jako kryteria, według których dokonano porównania narzędzi SikuliX oraz AutoIT v3 zostały przyjęte:

- czas wykonywania się skryptów przy uzyskaniu tego samego pokrycia,
- złożoność skryptu (liczba linii kodu),
- łatwość utrzymania,
- niezawodność skryptów przy wielokrotnym ich wykonaniu.

Powyższe kryteria wykorzystano do potwierdzenia tezy:
Nrzędzie AutoIT v3 jest bardziej efektywne w automatycznych testach GUI w porównaniu do SikuliX.

3.2. Platforma testowa

Tabela 1 przedstawia specyfikację maszyny, na której były uruchamiane testy.

Tabela 1. Specyfikacja maszyny testowej

Procesor	Intel Pentium T4400
Taktowanie procesora	2,20 GHz
Liczba rdzeni	2
RAM	8 GB
Rozdzielczość	1366 x 768
System operacyjny	Windows 7 Professional 64-bit SP1

4. Wyniki badań

4.1. Czas wykonywania się skryptów

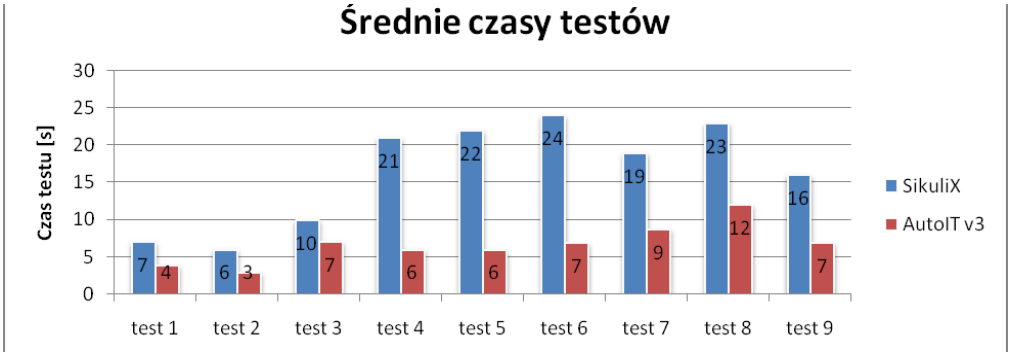
Na rysunku 1 zestawiono średnie czasy, jakie otrzymano dla poszczególnych testów w obu narzędziach. AutoIT okazało się najszybsze we wszystkich testach. Najszybszym testem okazał się test *Błędne logowanie*. Wykonywał się on w czasie 3 s., podczas gdy w SikuliX 6 s. Najdłużej wykonującym się skryptem w AutoIT był test *Rejestracja pacjenta*. Średni czas, jaki osiągnięto dla niego w tym narzędziu to 12 s. W SikuliX test ten okazał się drugim najwolniejszym testem, a trwał on 23 s. Natomiast najwolniejszym testem dla tego narzędzia okazał się test *Dodanie pacjenta*. Wykonywał się on sekundę dłużej niż skrypt wspomniany wcześniej, czyli 24 s.

Rysunek 2 prezentuje zestawienie czasów otrzymanych dla narzędzia SikuliX. Dla jednego z uruchomień testu *Dodanie pacjenta* otrzymano czas 33 s. Wynik ten okazał się największym czasem jaki osiągnięto spośród wszystkich uruchomień skryptów w SikuliX. Natomiast dla testu *Logowanie* oraz *Błędne logowanie* otrzymano w uruchomieniach najkrótszy czas, a wynosił on 4 s.

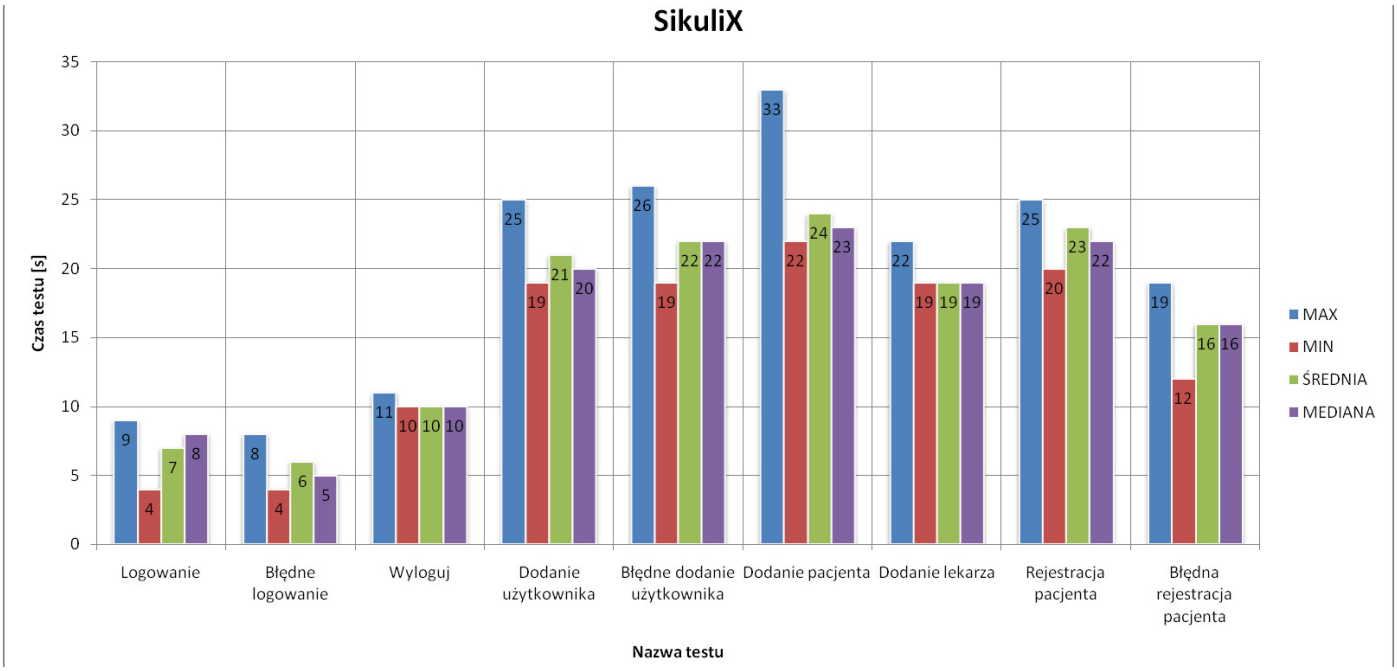
Na rysunku 3 przedstawiono zestawienie czasów otrzymanych w AutoIT. Dla testu *Rejestracja pacjenta* otrzymany czas podczas jednego z uruchomień skryptu okazał się największym wynikiem spośród wszystkich otrzymanych. Wynosił on 14 s. Najkrótszy czas uzyskano pośród wykonywania się testu *Błędne logowanie*. Wynik ten wynosi 2 s.

4.2. Złożoność skryptu

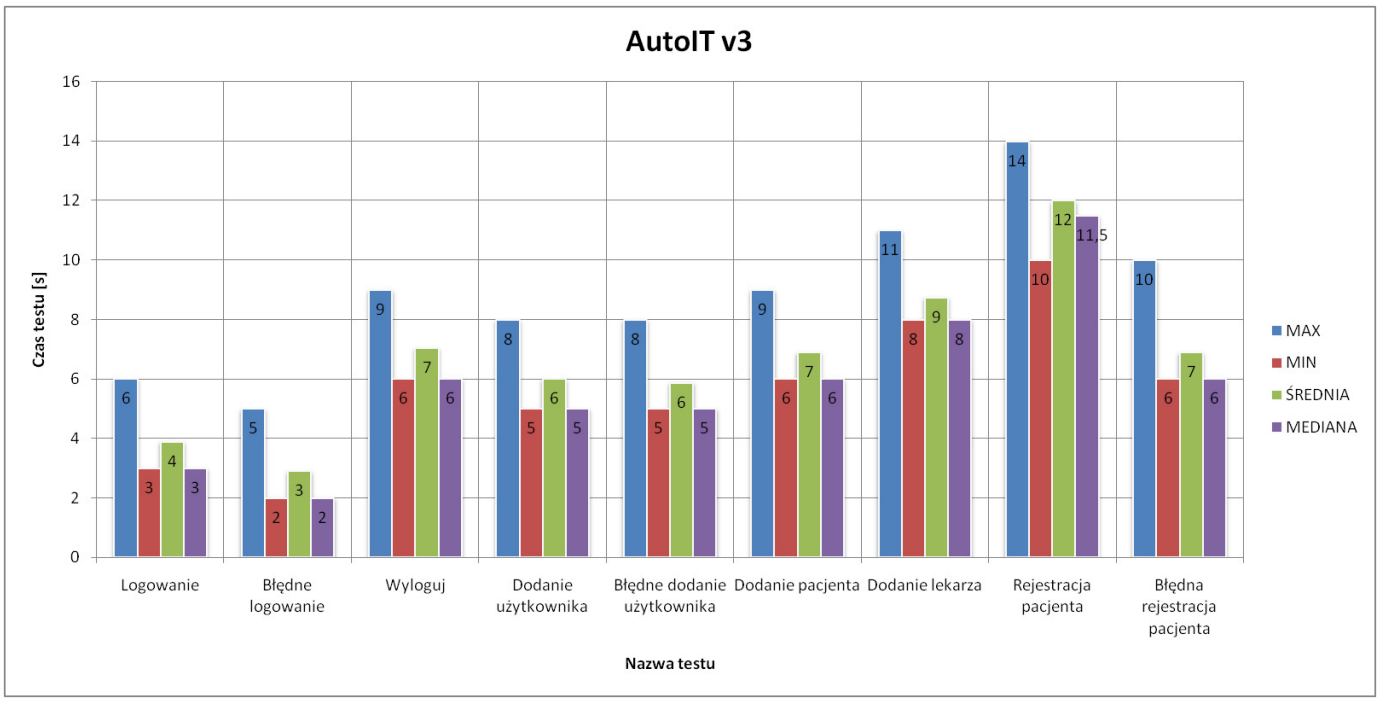
Na rysunku 4 przedstawiono zbiorcze zestawienie liczby linii kodu dla wszystkich testów napisanych w obu narzędziach. Testy napisane w SikuliX okazały się zajmować mniejszą liczbę linii kodu w porównaniu do AutoIT v3. Najmniejszą liczbę linii kodu w tym narzędziu otrzymano dla testu *Błędne logowanie*, a wynosiła ona 10 wierszy. Najdłuższym testem napisanym w SikuliX okazał się *Rejestracja pacjenta*. Liczył on 34 linie. Jednocześnie dla tego testu otrzymano największą różnicę w długości skryptu pomiędzy narzędziami. Test ten napisany w AutoIT zajmuje 59 linii, a więc różnica wynosi 25 wierszy. Najkrótsze testy w tym narzędziu to testy dotyczące logowania: *Logowanie* oraz *Błędne logowanie*, które zajmują odpowiednio 15 oraz 14 linii kodu. Są to jedyne testy, które pod względem złożoności są najbardziej zbliżone do testów napisanych w SikuliX.



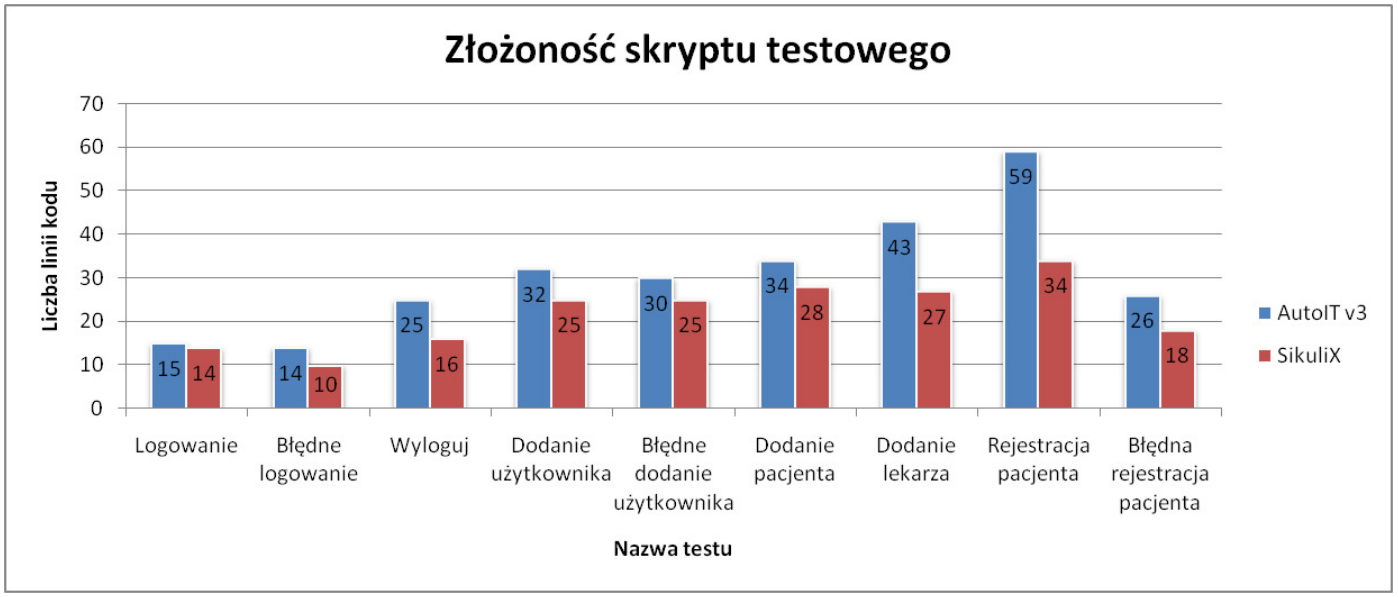
Rys.1. Średnie czasy testów w obu narzędziach



Rys. 2. Czasy otrzymane dla SikuliX



Rys. 3. Czasy otrzymane dla AutoIT



Rys. 4. Zestawieni liczby linii kodu skryptów testowych w obu narzędziach

4.3. Łatwość utrzymania

Następnym badanym aspektem była łatwość utrzymania skryptu testowego. Cecha ta jest niejako związana z samym zaprojektowaniem i sposobem napisania testu. Sam test musi być przemyślany, trzeba zaplanować, co dany test ma robić. Przy pisaniu testu warto koncentrować się na jak najprostszym sposobie, dzięki czemu wprowadzenie ewentualnych zmian w późniejszym życiu testu będzie łatwiejsze.

SikuliX opiera się na rozpoznawaniu obrazu, a w skryptach testowych wykorzystuje się zrzuty ekranu GUI. Testy w tym narzędziu są podatne na jakiekolwiek, nawet najmniejsze modyfikacje interfejsu użytkownika, zarówno w samym wyglądzie, jaki i rozmieszczeniu poszczególnych kontrolek. Wszystko to sprawia, że utrzymanie testów w tym narzędziu jest czasochłonne i mozolne. Dlatego warto zastanowić się tu, w jaki sposób można zaprojektować test. Jednym z rozwiązań jest wykonywanie tam gdzie to możliwe oddzielnych zrzutów ekranu dla poszczególnych elementów GUI, zamiast jednego, który uchwyci, np. całe okno, gdzie trzeba będzie wykonywać zmiany współrzędnych dla poszczególnych akcji. Wywoływanie innych skryptów z poziomu drugiego testu również wpłynie pozytywnie na łatwość utrzymania. Jeżeli skrypty wykonują jakąś wspólną część - warto ją zaimplementować jako oddzielny test, a wywoływać go w miejscach potrzebnych. Gdyby coś się zmieniło w tym skrypcie to potrzeba naniesienia zmian będzie tylko w jednym miejscu, a nie we wszystkich.
Testy napisane w AutoIT v3 wykorzystują symulację użycia myszy lub klawiatury komputerowej na elementach interfejsu użytkownika. Aby zdarzenie mogło zaistnieć narzędzie musi wiedzieć na jakim elemencie ma wykonać akcję. Identyfikacja elementów GUI odbywa się poprzez podanie jego identyfikatora, którym może być np.: nazwa elementu (np. nazwa okna), id kontrolki nadany na etapie kodowania, tekst widoczny na elemencie. Użycie w komendach bezpośrednio identyfikatora elementu, powoduje, że skrypt staje się nieczytelny oraz jest trudny w utrzymaniu. Niejednokrotnie w różnych skryptach identyfikatory będą się powielać, dlatego rozwiązaniem, które znacząco polepszy utrzymanie testów jest stworzenie oddzielnego pliku ze zmiennymi, do których zostaną przypisane identyfikatory elementów GUI użytych w skrypcie. Na rysunku 5 przedstawiono przykładowy plik ze zmiennymi.

```
1  $imie = "[NAME:imie_txtb]"
2  $nazwisko = "[NAME:nazwisko_txtb]"
3  $haslo2 = "[NAME:haslo_txtb]"
4  $potw_haslo = "[NAME:potwHaslo_txtb]"
5  $akceptuj = "[NAME:akceptuj_btn]"
6  $anuluj = "[NAME:anuluj_btn]"
7  $dodaj_uzyt = "[Title:Dodaj użytkownika]"
8  $potw_dod_uzyt = "[Title:Dodanie użytkownika]"
9  $ok = "[Text:OK]"
10 $msg_bledne_dane = "[Title:Niepoprawne dane]"
11 $err_prov = "[CLASS:WindowsForms10.Window.8.app.0.141b42a_r15_ad1; INSTANCE:1]"
```

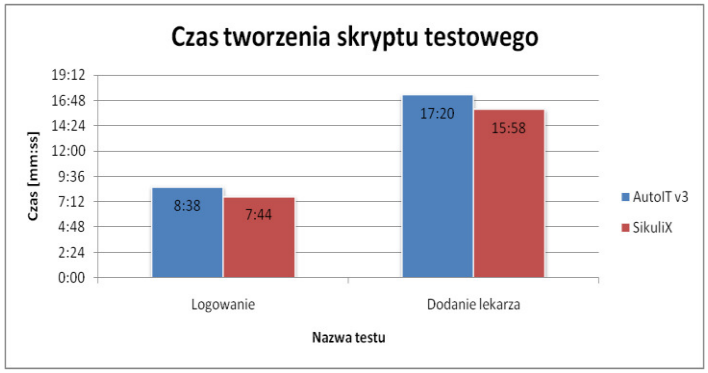
Rys. 5. Przykładowy plik ze zmiennymi

Na rysunku 6 przedstawiono czas, w jakim zostały napisane testy *Logowanie* oraz *Dodanie lekarza* celem zobrazowania czasochłonności. Czas poświęcony na tworzenie wybranych skryptów w obu narzędziach był zbliżony do siebie, jednak w narzędziu SikuliX testy powstawały szybciej. Napisanie skryptu testowego *Logowanie* w SikuliX zajęło autorowi 7 min i 44 s., a w AutoIT 8 min 38 s. Test w AutoIT powstawał 54 s. dłużej. Natomiast bardziej złożony test, jakim jest *Dodanie lekarza* wymagał ponad dwa razy tyle czasu. Napisanie go w SikuliX zajęło 15 min 58 s, podczas gdy w AutoIT 17 min 20 s.

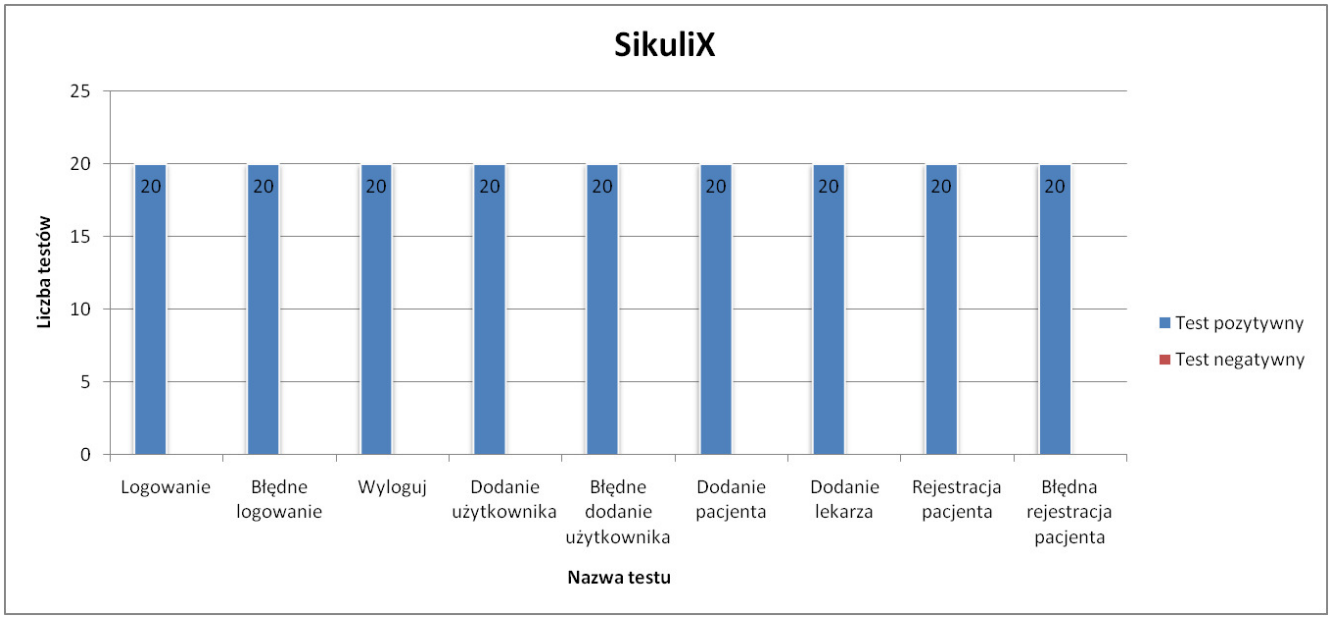
4.4. Niezawodność skryptów

W badanym aspekcie skoncentrowano się na liczbie testów, które zakończyły się powodzeniem. Każdy z testów został uruchomiony 20 razy, następnie zliczono liczbę uruchomień dających wynik pozytywny jak i negatywny. Na

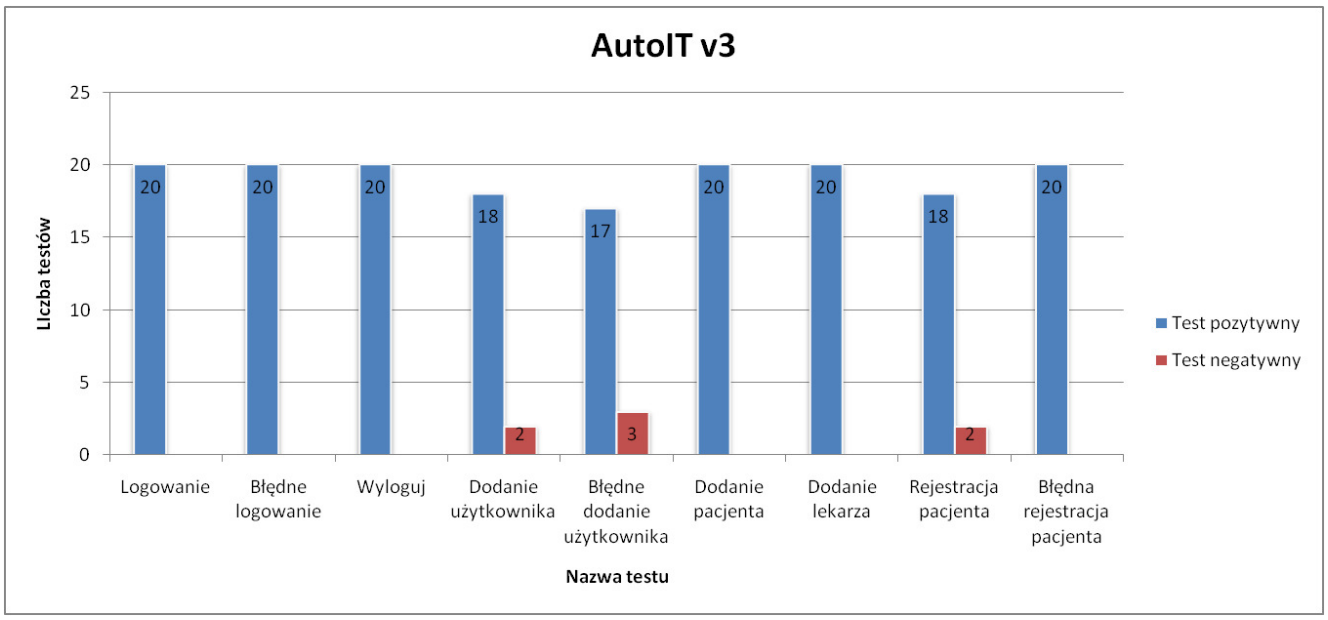
rysunku 7 oraz rysunku 8 przedstawiono wyniki dla poszczególnych narzędzi. Testy w SikuliX (rysunek 7) uzyskały 100% poprawności. Każde uruchomienie testu kończyło się wynikiem pozytywnym.
W przypadku testów napisanych w AutoIT v3 (rysunek 8) dla testów: *Logowanie*, *Błędne logowanie*, *Wyloguj*, *Dodanie pacjenta*, *Dodanie lekarza*, *Błędna rejestracja pacjenta* uzyskano przy każdym uruchomieniu wynik pozytywny. Natomiast dla testów *Dodanie użytkownika* oraz *Rejestracja pacjenta* otrzymano wynik pozytywny w 90%. *Błędne dodanie użytkownika* kończyło się największą liczbą niepowodzeń spośród wszystkich testów. Test ten trzy razy kończył się negatywnie, co daje 85% wyników pozytywnych.



Rys. 6. Czas pisania wybranych skryptów



Rys. 7. Niezawodność testów napisanych w SikuliX



Rys. 8. Niezawodność testów napisanych w AutoIT v3

5. Podsumowanie

Analizując otrzymane wyniki oraz odnosząc postawioną tezę do każdego z kryteriów oddzielnie można dostrzec, że teza znalazła potwierdzenie w kliku przyjętych kryteriach. Stąd oba narzędzia mają swoje słabe i mocne strony. AutoIT v3 okazało się narzędziem znacznie szybszym od SikuliX i to w niektórych testach kilkukrotnie. Jednakże testy w AutoIT były mniej niezawodne. W kilku przypadkach test kończył się niepowodzeniem. SikuliX pod względem niezawodności okazało się bezbłędne. Wszystkie uruchomienia poszczególnych testów dawały wynik pozytywny.

W aspekcie złożoności skryptów SikuliX również okazało się lepsze. Wszystkie skrypty w tym narzędziu były krótsze o klika, kilkanaście linii kodu. Różnica w głównej mierze wynika ze składni języka zastosowanego w narzędziach. Jednak analizując złożoność skryptów wraz z czasem ich wykonania można spostrzec, że liczba linii kodu nie ma znaczenia. Pomimo tego, że testy w AutoIT v3 były dłuższe to wykonywały się szybciej od SikuliX. Przykładem tu może być skrypt *Rejestracja pacjenta*. W AutoIT v3 zajmował 59 linii kodu, gdzie w SikuliX 34 linii. Jednak czas wykonywania był w przybliżeniu dwa razy krótszy na korzyść AutoIT.

Łatwość utrzymania wynika m.in. z samej konstrukcji narzędzi. Testy w AutoIT opierają się o identyfikatory elementów interfejsu użytkownika. Z racji, że zmieniają się one rzadko to testy w tym narzędziu są stabilniejsze od skryptów w SikuliX, które opierają się na rozpoznawaniu obrazu, a warstwa graficzna interfejsu użytkownika jest podatna na zmiany. Wprowadzanie ewentualnych zmian do skryptów tworzonych w tych narzędziach jest porównywalnie czasochłonne.

W ogólnym rozrachunku oraz przy badanej aplikacji lepszym rozwiązaniem jest oparcie testów GUI o narzędzie AutoIT. Aplikacja jest desktopowa na system Windows. Jest tu dostęp do identyfikatorów elementów GUI, a co za tym idzie skrypty będą mniej podatne na zmiany w warstwie graficznej. Testy będą szybsze i dodatkowo można je skompilować do plików exe.

Literatura

[1] Stowarzyszenie Jakości Systemów Informatycznych, Certyfikowany tester, Plan poziomu podstawowego, wersja 2011.1.1.

[2] IEEE 829 Standard for Software and System Test Documentation, IEEE Computer Society, 2008.

[3] Myers G. J., The Art of Software Testing, John Wiley & Sons, Inc., 2004.

[4] http://testerzy.pl/artykuly/definicja-testowania-oprogramowania-cz-1 [11.09.2017.].

[5] Roman A., Testowanie i jakość oprogramowania. Modele, techniki, narzędzia, Wydawnictwo Naukowe PWN, 2016.

[6] Black R., Advanced Software Testing Vol. 2, Guide to the ISTQB Advanced Certification as an Advanced Test Manager, Rock Nook Inc., 2011.

[7] Stowarzyszenie Jakości Systemów Informatycznych, Certyfikowany tester. Sylabus dla Poziomu Zaawansowanego. Kierownik Testów, werjsa 2012.

[8] International Software Testing Qualifications, Board Certified Tester Advanced Level Syllabus - Test Automation Engineer, 2016.

[9] http://sikulix.com/ [28.02.2018].

[10] https://media.readthedocs.org/pdf/sikulix-2014/latest/sikulix-2014.pdf.

[11] https://www.autoitscript.com/site/autoit/ [28.02.2018].

[12] https://opensourceforu.com/2017/01/autoit/ [dostęp 28.02.2018].