JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 28 (2023) 229–235
Received: 31 May 2023
Accepted: 20 June 2023

Comparative analysis of selected tools for test automation of web applications

Analiza porównawcza wybranych narzędzi do automatyzacji testów aplikacji webowych

Franciszek Wąsik*, Michał Pojęta*, Małgorzata Plechawska-Wójcik

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
The paper concerns a comparison of selected tools for conducting automated unit tests of web applications. It focuses on the testing of server and client parts. The aim of the paper is to answer the questions: which tools work best in creating automatic tests of server and client applications. In the context of the server part, unittest and pytest – libraries based on the Python language – are compared. The comparison of the client part tools is carried out in the context of testing applications programmed with the Angular framework and pairs Jasmine and Jest together. The research is based on the results of test execution times of the prepared test applications. Analogous tests were programmed with each tool and repeated several times to obtain reliable results. The research showed that among the tools for testing server applications, unittest is the more efficient, while in the case of tools for testing client applications, Jasmine shows higher performance.

Keywords: comparative analysis; automated testing; performance of test automation tools

Streszczenie
Artykuł dotyczy porównania wybranych narzędzi do przeprowadzania automatycznych testów jednostkowych aplikacji internetowych. Skupia się on na testach części serwerowej i klienckiej. Celem pracy jest uzyskanie odpowiedzi na pytania: które narzędzia sprawdzają się najlepiej w tworzeniu automatycznych testów aplikacji serwerowych oraz klienckich. W kontekście części serwerowej porównaniu zostały poddane unittest oraz pytest – narzędzia oparte o język Python. Porównanie narzędzi części klienckiej jest przeprowadzone w kontekście testowania aplikacji zaprogramowanych z wykorzystaniem szkieletu programistycznego Angular i zestawia ze sobą Jasmine oraz Jest. Badania oparto o wyniki czasów wykonania testów przygotowanych aplikacji testowych. Analogiczne testy zaprogramowano z użyciem każdego z narzędzi i wielokrotnie powtórzono w celu uzyskania wiarygodnych wyników. Badania wykazały, że spośród narzędzi do testowania aplikacji serwerowych wydajniejszym jest unittest, natomiast w przypadku narzędzi do testowania aplikacji klienckich, Jasmine legitymuje się wyższą wydajnością.

Słowa kluczowe: analiza porównawcza; testy automatyczne; wydajność narzędzi do automatyzacji testów

*Corresponding author

Email address: **franciszek.wasik@pollub.edu.pl** (F. Wąsik), **michal.pojeta@pollub.edu.pl** (M. Pojęta)

1. Wstęp

Testowanie oprogramowania jest bardzo ważnym etapem jego produkcji. To czy system zostanie odpowiednio przetestowany, w dużym stopniu wpływa na jego dalszy cykl życia. Przeprowadzenie testów ma na celu wyłonienie niezgodności w tworzonym oprogramowaniu i pozwala uniknąć wielu problemów związanych z jego wdrożeniem i utrzymaniem [1]. Aby wytworzyć wysokiej jakości oprogramowanie, proces testowania powinien być dla jego twórców kluczowym aspektem [2].

Niniejsza praca traktuje o narzędziach służących do automatyzacji testów aplikacji internetowych. Możliwość automatyzacji przynosi wiele korzyści dla procesu testowania oprogramowania. Nie tylko pozwala na jego przyspieszenie, ale także na zwiększenie liczby wykonywanych testów [3]. Inną korzyścią wynikającą ze stosowania automatycznych testów, jest zmniejszenie zasobów ludzkich koniecznych do wykonania testów [4]. Co więcej, automatyzacja wpływa na zmniejszenie wydatków wynikających z konieczności przeprowadzania testów oprogramowania, które mogą stanowić znaczną część kosztów jego wytworzenia [5-6].

Jednym z kluczowych aspektów, które wpływają na wdrożenie automatyzacji jest zmniejszenie czasu trwania procesu testowania [7]. Powstało wiele publikacji, które analizują korzyści płynące z zastosowania automatyzacji testów. Artykuł “The Impacts of Test Automation on Software’s Cost, Quality and Time to Market” [8] skupia się na zmierzeniu wpływu automatyzacji na koszty, jakość i czas wytworzenia oprogramowania. Przeprowadzone badania jednoznacznie wskazują, że zastosowanie automatyzacji daje pozytywne rezultaty we wszystkich tych aspektach.

Rynek oferuje liczne rozwiązania, które pozwalają przeprowadzać zautomatyzowane testy. Wybór odpowiedniego narzędzia może być kluczowy, aby proces testowania przebiegał efektywnie. Próba wyłonienia najlepszego była podejmowana w wielu publikacjach. Jedną z nich jest “Comparative Study of Automated

Testing Tools: Selenium, Quick Test Professional and Testcomplete" [9]. Artykuł ten skupia się na trzech popularnych narzędziach do automatyzacji testów: Selenium, Testcomplete oraz Quick Test Professional. Autorzy porównują powyższe technologie pod względem kosztów licencji, wsparcia, czy też wymaganych umiejętności programistycznych użytkownika. Jednakże ostatecznie stwierdzają, że wybór narzędzia powinien być dostosowany do indywidualnych potrzeb danego projektu.

W kontekście testowania aplikacji opartych o JavaScript, czy TypeScript jednymi z popularniejszych narzędzi są Jasmine i Jest. Jedna z publikacji [10] porusza temat obu z nich i wskazuje, że Jest staje się coraz bardziej popularny i przewyższa liczbą pobrań konkurencyjne narzędzie Jasmine. Autorzy opisują, że zastąpili w swojej aplikacji Jasmine przez Jest i wskazują na jego zalety. Wśród nich wymieniają między innymi prostotę konfiguracji i samowystarczalność, rozumianą przez brak konieczności korzystania z dodatkowych środowisk do uruchamiania testów. Ponadto zwracają uwagę na bezproblemowość migracji z jednego narzędzia na drugie, która wynika z tego, że składnia testów, które były zaprogramowane w Jasmine, była niemal całkowicie kompatybilna z Jest.

Publikacja o tytule "Web based Automation Testing and Tools" [11] porusza większość podstawowych aspektów testowania aplikacji. Artykuł opisuje różne rodzaje testów, takie jak jednostkowe, integracyjne, systemowe, czy akceptacyjne. Praca porównuje także narzędzia do testowania aplikacji webowych – Selenium oraz TestNG. Ponadto artykuł zwraca uwagę na zasadność testowania aplikacji w procesie tworzenia oprogramowania, a także na to jak istotny jest odpowiedni dobór narzędzi oraz technik pisania testów.

Najpopularniejszym standardem pisania testów jest TDD (ang. Test Driven Development). Publikacja "Overview of the Test Driven Development Research Projects and Experiments" [12] omawia wspomniany standard na wielu płaszczyznach. Analizuje ona wyniki eksperymentów, przeprowadzonych między innymi przez firmy Microsoft oraz IBM, które miały na celu zbadanie wpływu TDD na proces tworzenia oprogramowania. Badania wykazały, że nie można jednoznacznie stwierdzić, jakoby TDD było lepszym podejściem testowania od tradycyjnego. Wykazano, że technika TDD dostarcza lepsze pokrycie testów, natomiast nie potwierdzono by jej zastosowanie skracało czas tworzenia oprogramowania.

Rozszerzeniem techniki TDD jest BDD (ang. Behaviour Driven Development). BDD łączy technikę TDD, a więc podejście polegające na pisaniu testów automatycznych przy dodawaniu funkcjonalności w ramach ciągłej integracji (ang. Continuous Integration, CI) podczas wytwarzania oprogramowania, oraz testy akceptacyjne użytkowników. Głównym założeniem BDD jest testowanie oprogramowania przy jednoczesnym spełnianiu oczekiwań klientów. Pracą, która kompleksowo charakteryzuje BDD jest "A Study of the Characteristics of Behaviour Driven Development" [13]. Według autorów głównymi cechami BDD są: wszechobecny język (ang. Ubiquitous Language), iteracyjny proces dekompozycji (ang. Iterative Decomposition Process), opis z historią użytkownika i szablonami scenariuszy (ang. Plain Text Description with User Story and Scenario Templates), zautomatyzowane testy akceptacyjne z regułami mapowania (ang. Automated Acceptance Testing with Mapping Rules), czytelny kod specyfikacji zorientowanej (ang. Readable Behaviour Oriented Specification Code) na zachowanie oraz zachowanie napędzane w różnych fazach (ang. Behaviour Driven at Different Phases).

Python jest obecnie jednym z najpopularniejszych języków programowania na świecie w wielu dziedzinach IT – od uczenia maszynowego, przez sztuczną inteligencję oraz Big Data, na programowaniu aplikacji internetowych kończąc [14]. Python dostarcza wiele gotowych bibliotek do testowania aplikacji internetowych. Książka "Python Unit Test Automation – Practical Techniques for Python Developers and Testers" [15] przedstawia opisy standardów pisania testów przy użyciu języka Python – na przykład TDD. Ponadto zawiera ona przegląd bibliotek takich jak: doctest, unittest, nose, nose2 czy pytest.

Inną pozycją traktującą o testowaniu w kontekście języka Python jest książka "Testing Python: Applying Unit Testing, TDD, BDD and Acceptance Testing" [16] autorstwa David'a Sale'a. Podobnie jak w przypadku [15], opracowanie to zawiera opisy bibliotek takich jak unittest, nose, doctest, pytest czy nose2, wraz z przykładowymi testami. Ponadto opisuje metody używane w poszczególnych bibliotekach (głównie unittest). Dodatkowo publikacja ta omawia narzędzia do zarządzania pakietami języka Python (pip), tworzenia wirtualnego środowiska (virtualenv) oraz kontroli wersji (SVN, git), a także przykłady ich użycia. Co równie istotne, książka David'a Sale'a porusza też temat tworzenia testów akceptacyjnych, przy użyciu szkieletu Robot, a także automatyzacji procesu uruchamiania i zarządzania testami poprzez narzędzie Jenkins, oraz pokrycia testów aplikacji narzędziem Coverage.

Książką podobną do poprzednich dwóch pozycji jest "Python Testing: Beginner's Guide" [17], której autorem jest Daniel Arbuckle. Analogicznie do wcześniej wspomnianych publikacji, zawiera ona opisy bibliotek służących do testowania w języku Python, wraz z przykładami. Co więcej, książka ta rozwija temat testowania, poświęcając więcej uwagi testowaniu integracyjnemu oraz testowaniu systemu. Podobnie jak książka David'a Sale'a, rozwja ona temat testów akceptacyjnych, natomiast w tym przypadku autor użył narzędzia Twill.

Jedną z najpopularniejszych bibliotek służących do testowania aplikacji jest biblioteka pytest. Książka "Python Testing with pytest" [18] jest kompletnym przeglądem tego narzędzia. Pozycja ta zawiera opis, konfigurację oraz przykłady użycia biblioteki, wraz ze wszystkimi funkcjonalnościami, technikami testowania oraz wtyczkami. Każdy rozdział obejmuje inną część biblioteki pytest, wraz z przykładami użycia oraz ćwiczeniami. Inną pozycją traktującą o podobnej tematyce do

wspomnianej książki jest "pytest Quick Start Guide: Write better Python code with simple and maintainable tests" [19].

Mając na uwadze, że w niniejszej analizie zostały porównane narzędzia unittest oraz pytest, ciekawą publikacją jest artykuł "Assessing the migration of testing frameworks in the Python ecosystem" [20]. Jest to publikacja badająca migrację szeregu aplikacji, projektów czy też systemów z narzędzia unittest do biblioteki pytest. Z badania wynika, że ze 100 projektów 34% używa obu narzędzi. Jednocześnie potwierdzona została teza o zmianie narzędzia na pytest. Wśród głównych powodów autorzy wymieniają takie czynniki jak łatwiejsza składnia i utrzymanie, elastyczność oraz ponowne użycie poszczególnych komponentów. Jednak należy zauważyć, że w podanym badaniu brak jakichkolwiek porównań pod kątem wydajności obu narzędzi.

2. Cel i zakres pracy

Celem pracy jest przeanalizowanie oraz porównanie wybranych narzędzi służących do automatyzacji testów aplikacji internetowych. Praca składa się z dwóch części – pierwsza jest poświęcona narzędziom do wykonywania testów aplikacji serwerowych, natomiast druga, tym do testów aplikacji klienckich. Wybrane narzędzia zostały przetestowane pod względem wydajności. Dane badawcze pozyskano poprzez zebranie czasów wykonania analogicznych testów jednostkowych napisanych z wykorzystaniem poszczególnych narzędzi. Dla części traktującej o narzędziach do testowania aplikacji serwerowych zostało wykorzystane otwarto-źródłowe API. W części skupiającej się na narzędziach służących do testowania aplikacji klienckich, testy zostały napisane w oparciu o przygotowaną aplikację testową.

Tezy:

1. Jasmine jest wydajniejszym narzędziem do wykonywania testów jednostkowych dla aplikacji napisanych z użyciem szkieletu programistycznego Angular.
2. unittest jest wydajniejszym narzędziem do wykonywania testów jednostkowych w kontekście języka Python.

Pytania badawcze:

1. Czy można wyłonić wydajniejsze od domyślnego narzędzie do automatyzacji testów jednostkowych dla aplikacji napisanych z użyciem Angular?
2. Jakie narzędzie sprawdza się najlepiej w tworzeniu automatycznych testów jednostkowych aplikacji zaprogramowanych z użyciem Angular?
3. Jakie narzędzie sprawdza się najlepiej w tworzeniu testów automatycznych serwerowych aplikacji webowych?

3. Plan eksperymentu

Na plan eksperymentu badawczego składają się następujące etapy:

1. Przygotowanie aplikacji, na podstawie których zostanie przeprowadzona analiza porównawcza
    a. Przygotowanie aplikacji testowej dla narzędzi do testowania części serwerowej
    b. Przygotowanie aplikacji testowej dla narzędzi do testowania części klienckiej
2. Przygotowanie środowisk dla poszczególnych narzędzi służących do przeprowadzania testów
3. Utworzenie scenariuszy testowych
4. Przeprowadzenie testów oraz zgromadzenie otrzymanych wyników
    a. Wykonanie serii testów w celu uzyskania wiarygodnego średniego czasu ich trwania
    b. Zapisanie czasów wykonania poszczególnych testów jednostkowych
5. Analiza otrzymanych wyników
6. Wnioski

W celu otrzymania wiarygodnych wyników i stabilnych wartości statystycznych, testy zostały wykonane w kilku seriach. Dla aplikacji serwerowej zostały one powtórzone 10-krotnie. Ta liczba powtórzeń pozwoliła na uzyskanie precyzyjnych wyników. Testy aplikacji klienckiej, ze względu na duże różnice w czasie wykonania testów pomiędzy badanymi narzędziami, powtórzono 20-krotnie. Dla takiej liczby wyniki zostały uznane za wiarygodne.

Testy dla części serwerowej i klienckiej zostały wykonane na innych urządzeniach. Parametry sprzętu, na którym przygotowano środowisko testowe dla części serwerowej prezentuje Tabela 1, natomiast w Tabeli 2 przedstawiono parametry sprzętu wykorzystanego przy części klienckiej.

Tabela 1: Parametry sprzętu do testów części serwerowej

Parametr	**Wartość**
Procesor	Intel Core i7-9750H
Pamięć RAM	8GB
Dysk	256GB SSD
Karta graficzna	NVIDIA GeForce GTX 1650
System operacyjny	Windows 11 Home

Tabela 2: Parametry sprzętu do testów części klienckiej

Parametr	**Wartość**
Procesor	Intel Core i7-7700HQ
Pamięć RAM	16GB
Dysk	512GB SSD
Karta graficzna	NVIDIA GeForce GTX 1050 Ti
System operacyjny	Windows 11 Home

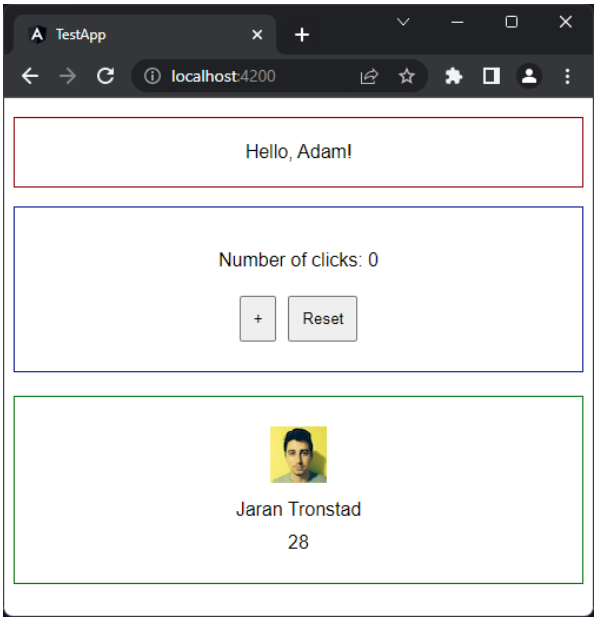
4. Aplikacje testowe

Na potrzeby przeprowadzenia badań zostały wykorzystane, specjalnie do tego celu przygotowane, aplikacje testowe. W oparciu o wspomniane aplikacje zaprogramowano testy jednostkowe z wykorzystaniem poddanych porównaniu narzędzi. Testy dla części serwerowej zostały zaprogramowane dla darmowego Api-Football w wersji 3.9.2. API to zwraca dane z zakresu tematyki piłki nożnej i pozwala pobrać historyczne dane dotyczące odbytych meczów, wyników, zespołów, stadionów, czy statystyk. Przykładowa odpowiedź z wykorzystanego API została przedstawiona na Rysunku 1.

```
    "get": "trophies",
    "parameters": {
        "player": "1100"
    },
    "errors": [],
    "results": 11,
    "paging": {
        "current": 1,
        "total": 1
    },
    "response": [
        {
            "league": "Community Shield",
            "country": "England",
            "season": "2022/2023",
            "place": "2nd Place"
        },
        {
            "league": "Bundesliga",
            "country": "Germany",
            "season": "2021/2022",
            "place": "2nd Place"
        },
```


Rysunek 1: Przykładowa odpowiedź z wykorzystanego API.

Dla części klienckiej zaprogramowano aplikację (Rysunek 2) z użyciem szkieletu Angular w wersji 15.2.6. Składa się ona z trzech niezależnych komponentów: GreetingComponent, ClickCounterComponent oraz UserComponent, które odpowiednio umożliwiają wyświetlenie powitania, obsługę licznika kliknięć oraz wyświetlenie danych użytkownika pobranych z zewnętrznego API.

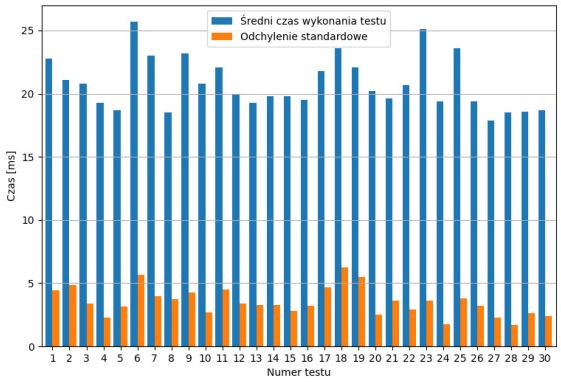


Rysunek 2: Kliencka aplikacja testowa.

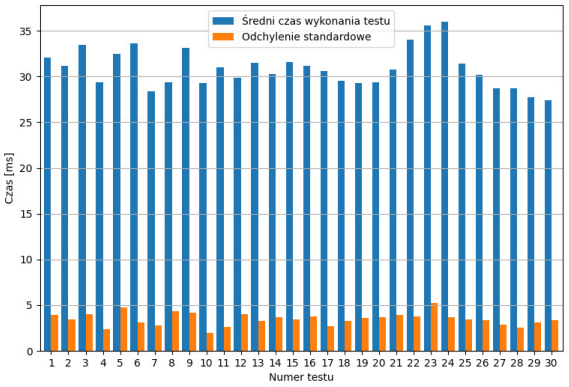
5. Wyniki badań

5.1. Wyniki badań narzędzi dla części serwerowej

Średnie czasy wykonania poszczególnych testów wraz z odchyleniami standardowymi dla biblioteki unittest zostały przedstawione na Rysunku 3, natomiast Rysunek 4 prezentuje te dane dla biblioteki pytest.



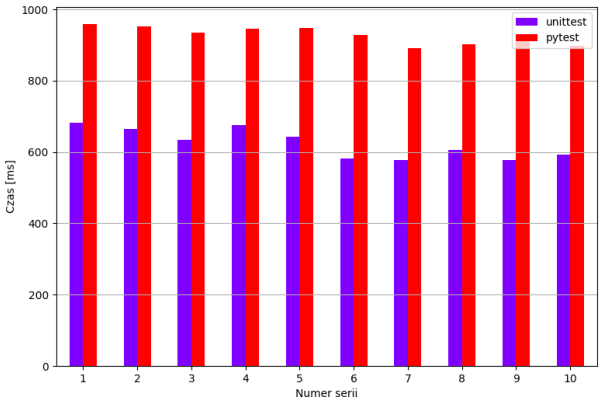
Rysunek 3: Średnie czasy wykonania oraz odchylenia standardowe dla testów napisanych z użyciem unittest.



Rysunek 4: Średnie czasy wykonania oraz odchylenia standardowe dla testów napisanych z użyciem pytest.

Analizując średnie czasy wykonania testów można stwierdzić, że testy napisane z wykorzystaniem unittest wykonywały się szybciej.

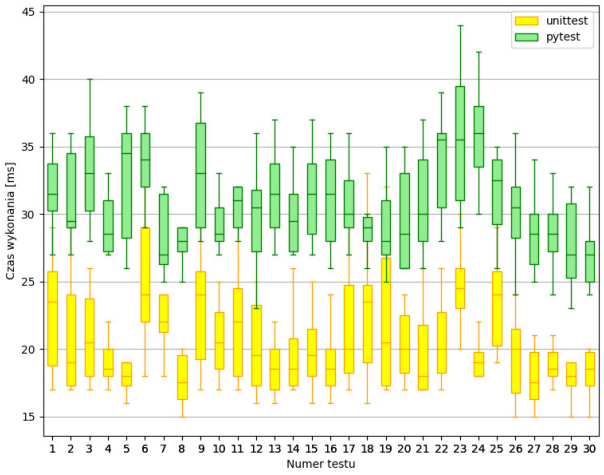
Wszystkie testy zostały powtórzone 10-krotnie, co pozwoliło uzyskać stabilne wartości statystyczne. Rysunek 5 prezentuje łączny czas wykonania wszystkich testów w kolejnych seriach. W każdej z nich unittest wypadł lepiej pod względem czasu wykonania, który średnio wynosił 624ms. Dla konkurencyjnej biblioteki pytest, średni czas potrzebny na wykonanie zestawu testów wyniósł 927ms.



Rysunek 5: Czasy wykonania wszystkich testów w poszczególnych seriach.

Na Rysunku 6 przedstawiono wykres pudełkowy przedstawiający czasy wykonania kolejnych testów dla poszczególnych bibliotek. Pokazuje on, że narzędzie

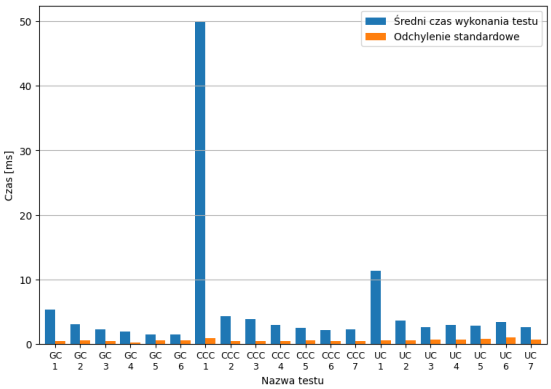
unittest jest bardziej wydajnym narzędziem niż pytest. Pomimo kilku wartości podobnych dla obu narzędzi, jak na przykład największy czas i najmniejszy czas wykonania danej serii dla przykładowego testu, można zauważyć trend, który jednoznacznie wskazuje narzędzie unittest jako bardziej wydajne. Ponadto na wykresie można zauważyć, że dla kilku testów (np. 12. oraz 23. test dla narzędzia pytest lub 18. test dla narzędzia unittest) różnica między maksymalnym a minimalnym czasem wykonania odbiega od różnic między tymi wartościami dla innych scenariuszy testowych.



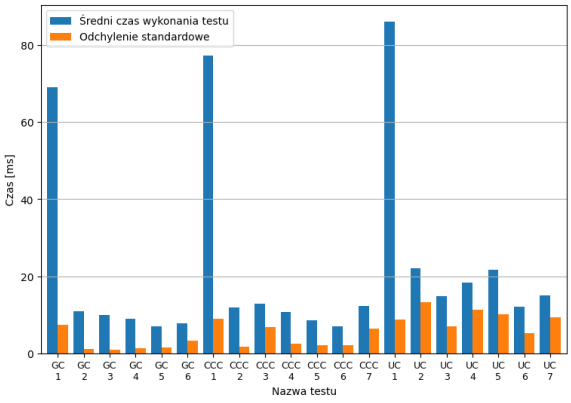
Rysunek 6: Rozkład czasu wykonania testów dla obu narzędzi.

5.2. Wyniki badań narzędzi dla części klienckiej

Średnie czasy wykonania oraz odchylenia standardowe poszczególnych testów dla narzędzi Jasmine oraz Jest przedstawiono odpowiednio na Rysunku 7 oraz 8. Wyniki zostały obliczone na podstawie 20 serii testów. Ta liczba powtórzeń pozwoliła na uzyskanie stabilnych wartości statystycznych i wiarygodnych wyników. Nazwy testów odpowiadają kolejnym testom napisanym dla poszczególnych komponentów aplikacji testowej. Testy *GC 1 – GC 6* są testami komponentu GreetingComponent, *CCC 1 – CCC 7* odpowiadają testom ClickCounterComponent, natomiast *UC 1 – UC 7* są testami UserComponent. Analizując średnie czasy wykonania poszczególnych testów można zauważyć, że w każdym przypadku narzędzie Jasmine było wielokrotnie szybsze od narzędzia Jest. Odchylenia standardowe są także znacznie mniejsze w przypadku Jasmine.

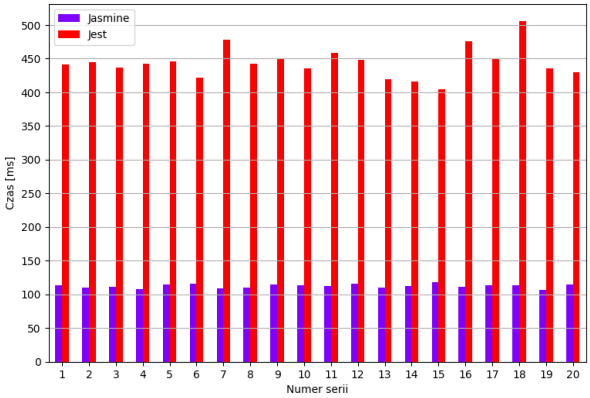


Rysunek 7: Średnie czasy wykonania oraz odchylenia standardowe dla testów napisanych z użyciem Jasmine.



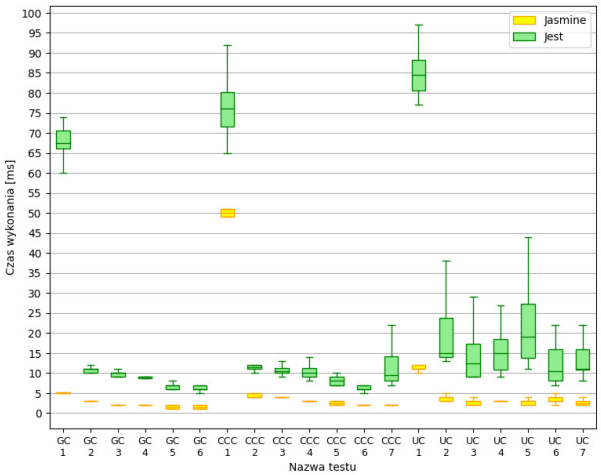
Rysunek 8: Średnie czasy wykonania oraz odchylenia standardowe dla testów napisanych z użyciem Jest.

Na Rysunku 9 przedstawiono czasy wykonania całego zestawu, składającego się z 20 testów, w poszczególnych seriach. Dla narzędzia Jasmine czasy te są bardzo podobne w każdej z serii i średnio wynoszą 113ms. W przypadku Jest można zaobserwować większą rozbieżność pomiędzy poszczególnymi wynikami, a średni czas potrzebny na wykonanie zestawu testów to 444ms.



Rysunek 9: Czasy wykonania wszystkich testów w poszczególnych seriach.

Rozkład czasu wykonania testów prezentuje wykres pudełkowy przedstawiony na Rysunku 10.



Rysunek 10: Rozkład czasu wykonania testów dla obu narzędzi.

Wykres ten potwierdza, że czasy wykonania testów dla narzędzia Jasmine są zauważalnie mniejsze. Ponadto pokazuje, że rozbieżność pomiędzy kolejnymi próbkami jest znacząco większa dla części testów wykonanych z wykorzystaniem narzędzia Jest. Można zatem stwierdzić, że jest ono mniej powtarzalne, jeśli chodzi o czas wykonywania się testów, od narzędzia Jasmine.

6. Wnioski

Niniejsza praca miała na celu porównanie ze sobą narzędzi służących do przeprowadzania automatycznych testów jednostkowych aplikacji internetowych. Porównane zostały narzędzia służące do testowania aplikacji serwerowych – unittest i pytest, oraz klienckich – Jasmine i Jest.

Przed przystąpieniem do realizacji eksperymentu zostały postawione tezy badawcze: *Jasmine jest wydajniejszym narzędziem do wykonywania testów jednostkowych dla aplikacji napisanych z użyciem szkieletu programistycznego Angular* oraz *unittest jest wydajniejszym narzędziem do wykonywania testów jednostkowych w kontekście języka Python*.

W celu przeprowadzenia eksperymentu badawczego przygotowano aplikacje testowe, w oparciu o które zostały zaprogramowane automatyczne testy jednostkowe. Czasy wykonania wspomnianych testów zostały zebrane i poddane analizie.

Po przeprowadzeniu eksperymentu badawczego, teza postawiona dla narzędzi do testów aplikacji klienckich znalazła swoje odzwierciedlenie w otrzymanych wynikach. Analiza czasów zebranych przez wielokrotne uruchomienie testów wykazała, że Jasmine jest znacznie wydajniejszym narzędziem niż Jest. Testy zaprogramowane z jego wykorzystaniem wykonywały się kilkukrotnie krócej, niż w przypadku konkurencyjnego narzędzia. Ponadto ich czasy wykonania były bardziej powtarzalne. Wyniki eksperymentu pozwalają odpowiedzieć na postawione pytania badawcze. Odpowiedź na pierwsze z nich – "*Czy można wyłonić wydajniejsze od domyślnego narzędzie do automatyzacji testów jednostkowych dla aplikacji napisanych z użyciem Angular?*" nie jest jednoznaczna. Wynika to z faktu, że z domyślnym narzędziem, którym jest Jasmine, zostało zestawione jedno konkurencyjne, a rynek oferuje też inne, mniej popularne narzędzia. Dlatego też w kontekście zestawionych w badaniu narzędzi odpowiedź na to pytanie jest negująca, jednakże nie można wykluczyć, że próba zestawienia Jasmine z innym narzędziem pozwoliłaby uzyskać odmienną odpowiedź. Drugie pytanie dotyczyło tego, które narzędzie sprawdza się najlepiej w tworzeniu automatycznych testów jednostkowych aplikacji zaprogramowanych z użyciem Angular. W tym przypadku badania wykazały, że pod względem wydajności lepiej sprawdza się Jasmine. Składnia testów dla obu narzędzi jest bardzo podobna, zatem konkurencyjny Jest nie wykazuje się widoczną przewagą w stosunku do domyślnego Jasmine. Można zatem wysnuć wniosek, że w kontekście aplikacji zaprogramowanych z wykorzystaniem szkieletu Angular, Jasmine sprawdza się lepiej.

Teza postawiona dla narzędzi do testów aplikacji serwerowych także została potwierdzona. Czasy wykonania testów stworzonych z wykorzystaniem biblioteki unittest były krótsze niż w przypadku pytest. Pomimo kilku omówionych przykładów, pokazujących duże różnice między maksymalnymi, a minimalnymi czasami wykonania poszczególnych testów, przeprowadzona analiza wyników jednoznacznie wskazuje narzędzie unittest jako bardziej wydajne. Nawiązując do postawionego pytania badawczego, dotyczącego tego jakie narzędzie sprawdza się najlepiej w tworzeniu testów automatycznych serwerowych aplikacji webowych, można stwierdzić, że wszystko zależy od przyjętych kryteriów. Jeśli najważniejszym kryterium jest wydajność, wtedy lepszym narzędziem jest biblioteka unittest. Jeśli głównymi czynnikami porównania jest prostota pisania testów, wtedy lepszym narzędziem jest pytest przez wzgląd na łatwiejszą składnię oraz obsługę testów.

Literatura

[1] M. E. Khan, F. Khan, Importance of Software Testing in Software Development Life Cycle, International Journal of Computer Science Issues (IJCSI) 11(2) (2014) 120-123.

[2] I. Bhatti, J. A. Siddiqi, A. Moiz, Z. A. Memon, Towards Ad hoc testing technique effectiveness in software testing life cycle, 2019 2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET) (2019) 1-6.

[3] M. A. Umar, C. Zhanfang, A Study of Automated Software Testing: Automation Tools and Frameworks, International Journal of Computer Science Engineering 6 (2019) 217-225.

[4] P. Kunte, D. Mane, Automation Testing of Web based application with Selenium and HP UFT (QTP), International Research Journal of Engineering and Technology (IRJET) 6 (2017) 2579-2583.

[5] D. Raghuvanshi, Introduction to Software Testing, International Journal of Trend in Scientific Research and Development (IJTSRD) 4(3) (2020) 797-800.

[6] V. Garousi, M. V. Mäntylä, When and what to automate in software testing? A multi-vocal literature review, Information and Software Technology 76 (2016) 92-117.

[7] D. Ateşoğulları, A. Mishra, Automation testing tools: A comparative view, International Journal on Information Technologies & Security 12(4) (2020) 63-76.

[8] D. Kumar, K. K. Mishra, The impacts of test automation on software's cost, quality and time to market, Procedia Computer Science 79 (2016) 8-15.

[9] H. Kaur, G. Gupta, Comparative Study of Automated Testing Tools: Selenium, Quick Test Professional and Testcomplete, Int. Journal of Engineering Research and Applications 3(5) (2013) 1739-1743.

[10] E. Pernice, C. Albiston, R. Beeler, E. Chou, C. Fry, M. Shor, J. Spears, D. Speck, A. Thakur, S. West, Application Development in the Face of Evolving Web Technologies at the National Ignition Facility, 17th Int. Conf. on Acc. and Large Exp. Physics Control Systems (2019) 1052-1056.

[11] M. Sharma, R. Angmo, Web based automation testing and tools, International Journal of Computer Science and Information Technologies 5(1) (2014) 908-912.

[12] A. Bulajic, S. Sambasivam, R. Stojic, Overview of the test driven development research projects and experiments, Proceedings of Informing Science & IT Education Conference (InSITE) (2012) 165-187.

[13] C. Solis, X. Wang, A study of the characteristics of behaviour driven development, 37th EUROMICRO conference on software engineering and advanced applications (2011) 383-387.

[14] A. Rawat, A Review on Python Programming, International Journal of Research in Engineering, Science and Management 3(12) (2020) 8-11.

[15] A. Pajankar, Python Unit Test Automation: Practical Techniques for Python Developers and Testers, Apress, Nashik, 2017.

[16] D. Sale, Testing Python: Applying Unit Testing, TDD, BDD and Acceptance Testing, John Wiley & Sons, Chichester, 2014.

[17] D. Arbuckle, Python Testing: Beginner's Guide, Packt Publishing Ltd, Birmingham, 2010.

[18] B. Okken, Python Testing with pytest, Pragmatic Bookshelf, Raleigh, 2022.

[19] B. Oliveira, pytest Quick Start Guide: Write better Python code with simple and maintainable tests, Packt Publishing Ltd., Birmingham, 2018.

[20] L. A. Barbosa, Assessing the migration of testing frameworks in the Python ecosystem, Master Thesis, Universidade Federal de Minas Gerais, Belo Horizonte, 2022.