JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 18 (2021) 7–14
Received: 26 October 2020
Accepted: 23 November 2020

Comparative analysis of solutions used in automated testing of Internet applications

Analiza porównawcza rozwiązań wykorzystywanych w testowaniu automatycznym aplikacji internetowych

Magdalena Psujek* , Aleksandra Radzik*, Grzegorz Kozieł

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

Abstract
This article is devoted to the comparison of solutions used in automatic testing of web applications. In order to carry out the analysis, test scenarios were created and tests were carried out using each of the tested programming frameworks (Selenium WebDriver, Cucumber, Ranorex, Robot Framework, Cypress, Unified Functional Testing, TestComplete and Katalon Studio). The study showed that there is no single best tool that meets all the requirements. Taking into account the analyzed aspects of effectiveness, the TestComplete program was the best, however, when choosing a solution, the team's skills and project specification should be taken into account.

Keywords: automated testing; testing frameworks

Streszczenie
Niniejszy artykuł został poświęcony porównaniu rozwiązań wykorzystywanych w testowaniu automatycznym aplikacji internetowych. W celu przeprowadzenia analizy stworzono scenariusze testowe oraz przeprowadzono testy z użyciem każdego z badanych narzędzi (Selenium WebDriver, Cucumber, Ranorex, Robot Framework, Cypress, Unified Functional Testing, TestComplete oraz Katalon Studio). W pracy wykazano, że nie istnieje jedno najlepsze rozwiązanie, spełniające wszystkie wymagania. Biorąc pod uwagę analizowane aspekty efektywności program TestComplete wypadł najlepiej, jednak przy wyborze rozwiązania należy wziąć pod uwagę umiejętności zespołu oraz specyfikację projektu.

Słowa kluczowe: testowanie automatyczne; frameworki testujące

*Corresponding author
Email addresses: m.e.psujek@gmail.com (M. Psujek), aradzik96@gmail.com (A. Radzik)

1. Wstęp

Oprogramowanie zawierające błędy nie powinno być wprowadzane do użytku komercyjnego. Nieprzetestowany produkt może spowodować straty finansowe a nawet być przyczyną choroby lub śmierci wielu osób [1].

W niniejszej pracy porównywane zostały narzędzia służące do tworzenia testów automatycznych. Takie testy wykonywane są gdy konieczne jest wielokrotne powtarzanie danego przypadku lub podczas symulowania dużego obciążenia. Automatyzacji testów najczęściej poddawane są systemy, które nie ulegają zmianom lub takie, w których wprowadzane zmiany są niewielkie i nie występują często. Testy automatyczne dają wyniki znacznie szybciej niż testy manualne, które przeprowadzone muszą być ręcznie przez pracownika [2]. W przeciwieństwie do maszyny, człowiek może popełnić błędy przy wprowadzaniu danych lub wykonać niepoprawne operacje systemowe [3]. Test przeprowadzany manualnie za każdym razem będzie dawał inny wynik chociażby, ze względu na czas jego wykonania. Zaletą testów automatycznych jest to, że wykonywane są zawsze z tą samą precyzją.

2. Analiza literatury

Powstało wiele dzieł podejmujących temat testowania automatycznego [4], których podejście zostanie przedstawione w niniejszym rozdziale. Na początku jednak należy wyjaśnić czym jest testowanie oprogramowania w jakim celu się stosuje oraz jakie są jego rodzaje.

Zgodnie z “Sylabusem poziomu podstawowego ISTQB®” [5] testowanie oprogramowania daje możliwość oceny jego jakości a także umożliwia zmniejszenie ryzyka wystąpienia niechcianych incydentów czy awarii w trakcie użytkowania systemu. W zależności od poziomu testu celem może być zlokalizowanie jak największej liczby defektów lub sprawdzenie czy system spełnia oczekiwania i został stworzony zgodnie z wymaganiami.

Testowanie oprogramowania można podzielić na dwie główne kategorie: manualne oraz automatyczne [6]. W artykule “Introduction to Software Testing” [7] autorstwa Durgesh Raghuvanshi, poruszony został temat testowania automatycznego. Zauważono w nim, że testowanie oprogramowania pochłania 50% kosztów jego wytwarzania. Zwrócono uwagę, że wprowadzenie automatyzacji pozwala na zmniejszenie zapotrzebowania na zasoby ludzkie, zwiększa dokładność, co za tym idzie zmniejsza ryzyko popełnienia błędu w trakcie przeprowadzania testu oraz znacznie skraca czas potrzebny na wykonanie testu.

Niniejsza praca traktuje o testach automatycznych funkcjonalnych. Testy funkcjonalne to proces, w ramach którego QA Specialist (Quality Assurance

Specialist- ang. Specjalista ds. Zapewniania Jakości) określa, czy oprogramowanie działa zgodnie z wcześniej ustalonymi wymaganiami. Wykorzystuje techniki testowania czarnej skrzynki, w których tester nie ma wiedzy o wewnętrznej logice systemu. Testy funkcjonalne dotyczą wyłącznie sprawdzania poprawności, czy system działa zgodnie z przeznaczeniem [5]. Niepowodzenie przeprowadzenia testu (uzyskanie niepoprawnego wyniku) może być spowodowane błędami w implementacji kontrolera czy tez modelu. Dlatego też analizowanie rodzajów błędów jakie są w stanie wykryć testy funkcjonalne mija się z celem. Tu sprawdza się czy wszystkie elementy systemu jako całość poprawnie współpracują i pozwalają na uzyskanie właściwego efektu działania aplikacji obsługiwanej przez interfejs graficzny dostępny przez przeglądarkę. Termin testy funkcjonalne występuje również w "Sylabusie poziomu podstawowego ISTQB®" [8], w którym wyjaśniono, że testy funkcjonalne to proces uwzględniający zachowanie systemu. W sylabusie przedstawiono zalety i ryzyka automatyzacji testów. Ze względu na to, że temat zalet został już poruszony w poprzednich akapitach rozważone zostaną zagrożenia takiego rozwiązania. Zgodnie z materiałem źródłowym należą do nich:

- oczekiwanie, że wybrane narzędzie będzie proste w obsłudze oraz spełni nierealistyczne wymagania;
- złe oszacowanie liczby godzin i kosztów wdrożenia narzędzia;
- złe oszacowanie liczby koniecznych działań, które miałoby zapewnić trwałe i znaczące korzyści z automatyzacji;
- całkowite zastąpienie testów manualnych automatycznymi;
- wycofanie narzędzia z rynku lub sprzedaż innemu dostawcy;
- brak współpracy z nowymi technologiami.

W zasobach portalu www.researchgate.net znaleziono artykuł "A Study of Automated Software Testing:Automation Tools and Frameworks" [9] autorstwa Mubarak Albarka Umar oraz Chen Zhanfang. Autorzy porównują w nim cztery narzędzia analizowane również w niniejszej pracy. Należą do nich: Katalon Studio, Unified Functional Testing, Selenium oraz TestComplete. Wymieniono wady i zalety każdego z rozwiązań. Jednak autorzy zaznaczają, że nie ma dobrego lub złego narzędzia do automatyzacji. Każde ma inne przeznaczenie i wymaga innego nakładu pracy. Wybór odpowiedniego narzędzia powinien zależeć od charakteru oprogramowania.

Następnym artykułem traktującym o testowaniu automatycznym jest "Analiza porównawcza rozwiązań wykorzystywanych w testowaniu automatycznym" [10] autorstwa: Agnieszka Dorota Wac, Tomasz Kamil Watras, Grzegorz Kozieł. W pracy zostały porównane SeleniumIDE, TestComplete, SahiPro oraz Katalon Studio. Twórcy analizowani wybrane narzędzia między innymi pod kątem procesu tworzenia testów oraz prędkości ich wykonywania. Na podstawie przeprowadzonych badań nie wybrano najlepszego i uniwersalnego programu służącego do wykonywania automatycznych testów.

Szybkość wykonywania testów automatycznych jest jednym z kluczowych aspektów testowania. Testy automatyczne są tworzone aby tester w dowolnym momencie pracy mógł w szybki sposób zweryfikować poprawność swojej pracy i znaleźć błędy. Źródła wskazują, że testowanie powinno się odbywać jak najwcześniej w celu zredukowania kosztów naprawy błędów [11]. Wiąże się to z tym, że programista powinien testować kod wielokrotnie podczas jego tworzenia. Czas wykonania testów jest szczególnie istotny w przypadku testów regresyjnych, gdzie badany jest wpływ wprowadzonych zmian na wiele modułów kodu - tu szczególnie istotne jest szybkie wykonywanie testów pozwalające na uzyskanie wyników w rozsądnym czasie [12]. Ze względu na przytoczone argumenty w artykule skupiono się na aspektach czasowych wykonania testów.

3. Badane rozwiązania

Technologie internetowe, tak jak i inne dziedziny informatyki, rozwijają się w bardzo szybkim tempie [13]. Z tego względu zwiększa się konieczność tworzenia coraz lepszych i wydajniejszych sposobów ich testowania. Obecnie udostępniono wiele narzędzi służących do tworzenia automatycznych testów funkcjonalnych. W niniejszej pracy przebadano jedne z najpopularniejszych używanych przez testerów. Należą do nich: Selenium WebDriver, Cucumber, Ranorex, Robot Framework, Cypress, Unified Functional Testing (UFT), TestComplete oraz Katalon Studio[14-15].

4. Plan badań

Aby uzyskać jak najbardziej rzetelne porównanie, zdecydowano się na przeprowadzenie badań w formie testów z użyciem każdego z narzędzi wymienionych w punkcie trzecim niniejszego artykułu.

4.1. Metoda badań

Przygotowane zostały cztery scenariusze testowe. Po dwa dla każdej z testowanych stron. Scenariusze miały różną długość, aby sprawdzić które narzędzie lepiej radzi sobie z danym rodzajem testu. Dodatkowo sprawdzono czy testy wykrywają awarię na stronie. Awaria była symulowana na stworzonej na potrzeby tej pracy platformie. Za pomocą każdego z narzędzi został przeprowadzony ten sam zestaw testów w takich samych warunkach, z użyciem tej samej przeglądarki, platformy, dodatkowego oprogramowania, komputera.

4.2. Strona Moodle Pollub

Dla strony moodle1.pollub.pl utworzone zostały dwa scenariusze. Jednym z nich jest scenariusz zmieniający język na stronie i sprawdzający czy zmiana przebiegła poprawnie. Drugi test sprawdza czy wiadomości wysyłane do innych użytkowników są przez nich odbierane, następnie czy wiadomość jest usuwana. Test zakończony zostaje pozytywnie jeśli każda z tych czynności zostanie wykonana a użytkownik po dokonaniu zmian wylogowany.

4.3. Własna strona

Strona stworzona na potrzeby tej pracy to prosty sklep internetowy. Umożliwia rejestrację i logowanie użytkowników, dodanie do koszyka oraz zakup produktów po wcześniejszym zalogowaniu. Po zebraniu odpowiedniej liczby wyników testów, na stronie zasymulowano awarię.

5. Scenariusze testowe

Przygotowano cztery zestawy testowe:

- scenariusz testowy nr 1: Test zmiany języka Moodle Pollub, przypadek testowy 1.1: Test zmiany języka Moodle Pollub na język angielski,
- scenariusz testowy nr 2: Wysłanie wiadomości Moodle Pollub, przypadek testowy 2.1: Wysłanie wiadomości Moodle Pollub z konta o loginie 83821, na konto o loginie 83818,
- scenariusz testowy nr 3: Przegląd dostępnych produktów, przypadek testowy 3.1: Sprawdzenie dostępności wybranego produktu,
- scenariusz testowy nr 4: Sprawdzenie funkcjonalności koszyka, przypadek testowy 4.1: Dodawanie i usuwanie produktów z koszyka.

Przykładowo przedstawiono jeden z nich. Scenariusz przedstawiony w tabeli 1 oraz przypadek testowy przedstawiony w tabeli 2 zostały wykorzystane dwukrotnie: w pierwszej kolejności na sprawnej stronie oraz podczas symulacji awarii, która polegała na dezaktywowaniu przycisku w menu strony.

Tabela 1: Scenariusz testowy nr 4.

Nazwa	Sprawdzenie funkcjonalności koszyka
Cel	Celem przeprowadzenia scenariusza testowego jest weryfikacja funkcjonalności dotyczącej działania koszyka: dodawanie i usuwanie produktu a także rejestracja nowego użytkownika oraz usunięcie konta
Typ	Test funkcjonalny
Czynności przygotowawcze	Sprawdzenie posiadania przeglądarki internetowej, sprawdzenie posiadania aktualnej aplikacji. Uruchomienie serwera bazy danych
Czynności końcowe	Wyłączenie przeglądarki. Zapis wyników do bazy danych

Tabela 2: Przypadek testowy 4.1

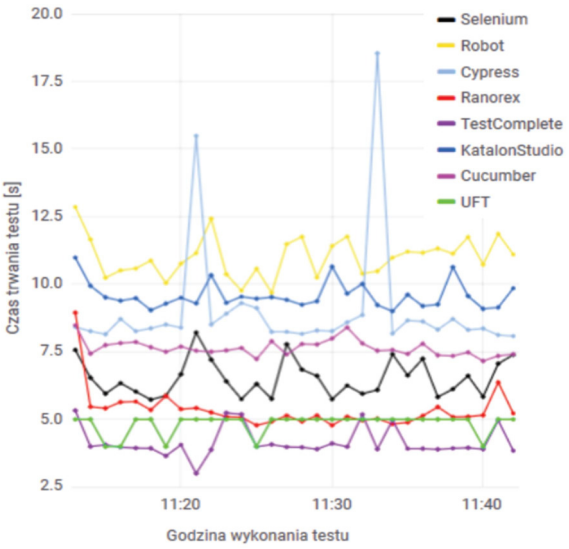
Nazwa	Dodawanie i usuwanie produktów z koszyka
Środowisko	Przeglądarka: Google Chrome Wersja 83.0.4103.116
Warunek wstępny	Włączona przeglądarka
Kroki	1. Otwórz stronę sklepu umieszczonego na localhost 2. Kliknij zarejestruj się 3. Wpisz imię “Test” 4. Wpisz nazwisko “Test” 5. Wpisz login “test1” 6. Wpisz miasto “Lublin” 7. Wpisz kod pocztowy „20-240” 8. Wpisz email “test@test.pl” 9. Wpisz hasło “testtest1” 10. Zaloguj się do sklepu internetowego 11. Przejdź do zakładki produkty 12. Wyszukaj produkt “Berberys” 13. Dodaj produkt do koszyka 14. Usuń produkt z koszyka 15. Kliknij wyświetl profil 16. Usuń konto
Oczekiwany rezultat	Poprawne dodanie do koszyka przez nowo zarejestrowanego klienta
Warunki końcowe	Produkt usunięty z koszyka, brak użytkownika w bazie

6. Rezultaty testów

W tym rozdziale przedstawione zostaną rezultaty przeprowadzonych testów. Dla każdego przypadku oraz narzędzia zrealizowano 30 testów. Przyjęto, że ze względu na powtarzalność czasów taka liczba powtórzeń wystarczy aby dać rzetelny średni wynik dla poszczególnych rozwiązań.

6.1. Test zmiany języka strony

Test zmiany języka strony został stworzony na podstawie scenariusza testowego nr 1 przypadku testowego 1.1 o nazwie "Test zmiany języka Moodle Pollub na język angielski".



Rysunek 1: Wykres czasu trwania wszystkich testów, dla przypadku testowego 1.1 .

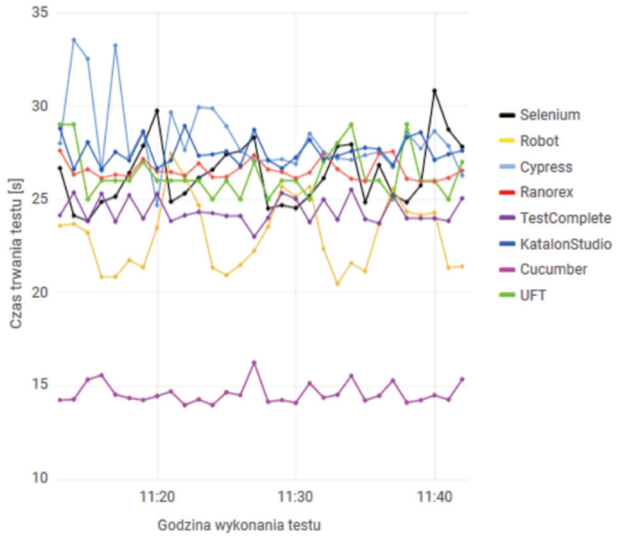
Rysunek 1 przedstawia wykres zbiorczy czasu trwania wszystkich testów, przeprowadzonych na podstawie przypadku 1.1 dla wykorzystanych narzędzi do testowania automatycznego. Z przedstawionego wykresu można odczytać, że najlepiej wypadło narzędzie TestComplete uzyskując najniższe czasy wykonania pojedynczego testu. Największy czas trwania odnotowano w przypadku framewoka Robot. Na wykresie można zauważyć również, że narzędzie Cypress posiada dwa duże odchyły od normy spowodowane przez niepoprawny wynik testu. Średni oraz całkowity czas trwania testów przedstawiono w tabeli 3.

Tabela 3: Wykaz najszybszych narzędzi.

Pozycja	Nazwa	Średni czas (s)	Całkowity czas (s)
1	TestComplete	4.15	124.8
2	UFT	4.83	145.2
3	Ranorex	5.35	160.8
4	Selenium	6.53	195.6
5	Cucumber	7.64	229.2
6	Cypress	9.03	271.2
7	Katalon	9.58	297.0
8	Robot	11.01	330.0

6.2. Test wysłania wiadomości strony

Test wysłania wiadomości dotyczy przypadku testowego 2.1 o nazwie "Wysłanie wiadomości Moodle Pollub z konta o loginie 83821, na konto o loginie 83818".



Rysunek 2: Wykres czasu trwania wszystkich testów, dla przypadku testowego 2.1.

Na rysunku 2 przedstawiony został zbiorczy wykres z wynikami długości trwania testów, przeprowadzonych na podstawie przypadku 2.1, każdego z narzędzi. Na wykresie od razu można dostrzec jeden wyróżniający się framework. Jest to Cucumber, uzyskał on najniższe czasy spośród wszystkich testowanych narzędzi. Jako drugie i trzecie - Robot Framework oraz TestComplete. Narzędziem, które najwolniej przeprowadziło test okazał się Cypress. Średni oraz całkowity czas trwania testów przedstawiono w tabeli 4.

Tabela 4: Wykaz najszybszych narzędzi.

Pozycja	Nazwa	Średni czas (s)	Całkowity czas (s)
1	Cucumber	14.61	438.6
2	Robot	23.11	693.6
3	TestComplete	24.36	730.8
4	Selenium	26.35	790.8
5	UFT	26.37	790.8
6	Ranorex	26.54	796.2
7	Katalon Studio	27.54	826.2
8	Cypress	28.24	847.2

6.3. Test sprawdzenia dostępności produktu

Kolejnym przypadkiem testowym na podstawie którego, analizowano czasy działania różnych narzędzi jest przypadek 3.1 o nazwie "Sprawdzenie dostępności wybranego produktu". Rysunek 3 przestawia wykres czasu trwania wszystkich testów, przeprowadzonych na podstawie przypadku 3.1, dla każdego z narzędzi.

Rysunek 3: Wykres czasu trwania wszystkich testów, dla przypadku testowego 3.1.

Widocznym jest, iż narzędzie Selenium uzyskało najlepszy czasu trwania testu. Zbliżone czasy do najszybszego z narzędzi odnotowano dla programów TestComplete oraz Cypress. Programem, który wykonał testy najwolniej okazał się Katalon. Średni oraz całkowity czas trwania testów przedstawiono w tabeli 5.

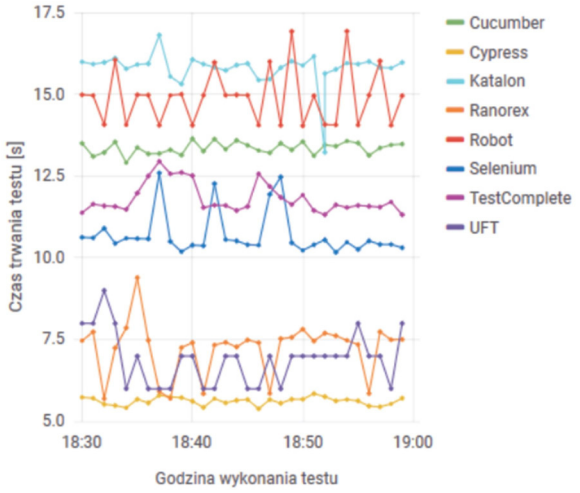
Tabela 5: Wykaz najszybszych narzędzi.

Pozycja	Nazwa	Średni czas (s)	Całkowity czas (s)
1	Selenium	1.7	49.8
2	TestComplete	1.7	51.0
3	Cypress	1.8	55,2
4	UFT	3.1	94.2
5	Ranorex	3.4	103.2
6	Cucumber	5.8	174.0
7	Robot	6.1	180.0
8	Katalon	7.4	222.0

6.4. Test funkcjonalności koszyka

Ostatnim przygotowanym przypadkiem testowym jest przypadek 4.1 o nazwie "Dodawanie i usuwanie produktów z koszyka". Przypadek ten przetestowano po trzydzieści razy dla każdego narzędzia.

Wyniki przeprowadzonych testów przedstawione zostały zaprezentowanie na rysunku 4. Średni oraz całkowity czas trwania testów przedstawiono w tabeli 6.



Rysunek 4: Wykres czasu trwania wszystkich testów, dla przypadku testowego 4.1.

Tabela 6: Wykaz najszybszych narzędzi.

Pozycja	Nazwa	Średni czas (s)	Całkowity czas (s)
1	Cypress	5.6	168.0
2	UFT	6.9	210.0
3	Ranorex	7.2	216,0
4	Selenium	11.0	324.0
5	TestComplete	11.8	354.0
6	Cucumber	13.0	402.0
7	Robot	15.0	450.0
8	Katalon	16.0	492.0

6.5. Symulacja awarii

Jako ostatni etap testowania narzędzi zasymulowano awarię poprzez wyłączenie funkcjonalności jednego z odnośnika na pasku menu. W ten sposób sprawdzono jak szybko narzędzia radzą sobie z wykryciem niepoprawności zaistniałej na stronie. Rysunek 5 przedstawia czasy trwania wszystkich testów. Najgorzej w tej próbie wypadł program Ranorex, każdy z testów wynosił 60s, jest to związane z tym, że maksymalny czas szukania elementu na stronie wynosi minutę, jest to czas ustawiony systemowo.

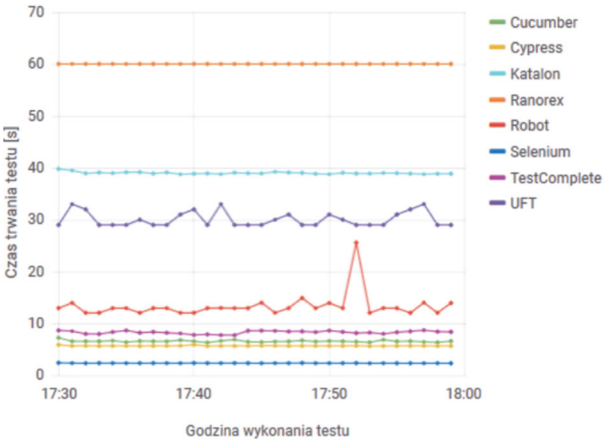
Średni oraz całkowity czas trwania testów przedstawiono w tabeli 7.

Tabela 7: Wykaz najszybszych narzędzi.

Pozycja	Nazwa	Średni czas (s)	Całkowity czas (min)
1	Selenium	2.4	70.8
2	Cypress	5.7	174.0
3	Cucumber	6.6	198.0
4	TestComplete	8.3	252.0
5	Robot	14.0	426.0
6	UFT	30.0	900.0
7	Katalon	39.0	1170.0
8	Ranorex	60.0	1800.0

7. Porównanie narzędzi

Tematem pracy było porównanie efektywności rozwiązań wykorzystywanych w testowaniu automatycznym aplikacji internetowych. Wybrano następujące kategorie, w których postanowiono przeanalizować rozwiązania zawarte w pracy: dostępność dokumentacji i wsparcie społeczności, poziom trudności konfiguracji oraz obsługi narzędzia, szybkość wykonania testu oraz prezentacja wyników.



Rysunek 5: Wykres czasu trwania wszystkich testów w przypadku awarii.

7.1. Dostępność dokumentacji

Ważnym elementem w trakcie korzystania z danego narzędzia jest dobrze i zrozumiale przygotowana dokumentacja. Postanowiono przeanalizować dokumentację rozwiązań pod kątem:

1. Dostępności (0-1),
2. Przejrzystości (0-2),
3. Łatwości w kontakcie i możliwości wsparcia społeczności tworzącej i korzystającej z danego narzędzia (0-4),
4. Częstotliwości aktualizacji (0-2),
5. Łatwości wyszukiwania (0-2).

Aby wybrać najlepsze narzędzie pod wskazanym aspektem postanowiono nadać odpowiednio punkty, przedstawione w nawiasach, dotyczące danej kategorii, ocena 0 oznacza brak, zaś najwyższa- możliwe najlepsze rozwiązanie w danej kategorii (tabela 8). Punkty przyznane w tabeli 8 zostały przydzielone na podstawie subiektywnych opinii autorów pracy dotyczących prezentacji i łatwości w korzystaniu z dokumentacji, ilości wątków w serwisie https://stackoverflow.com/, wprowadzanych zmian dotyczących konkretnego narzędzia w serwisie https://github.com/.

Tabela 8: Dostępność dokumentacji oraz wsparcie dostępności.

Nazwa	Numer kategorii					Suma
	1	2	3	4	5	
Cucumber	1	2	3	2	2	10
Cypress	1	2	2	2	2	9
Katalon	1	2	1	2	2	8
Ranorex	1	1	0	2	1	5
Robot Framework	1	1	3	2	1	8
Selenium	1	2	4	2	2	11
TestComplete	1	2	0	2	2	7
UFT	1	1	2	0	2	6

Na podstawie powstałej tabeli, można zauważyć, że większość z firm produkujących narzędzia służące do testowania automatycznego dba o dobrze dostępną oraz zrozumiałą dokumentację. Udostępnia również wsparcie poprzez utworzone grupy oraz fora internetowe.

7.2. Poziom trudności konfiguracji oraz obsługi narzędzia

Na podstawie rozdziału szóstego niniejszej pracy postanowiono podsumować poziom skomplikowania każdego z narzędzi.

Tabela 9: Charakterystyka poszczególnych narzędzi.

Nazwa	Instalacja	Dodatkowe oprogramowanie	Interfejs
Cucumber	dołączenie biblioteki	kompilator wybranego języka	brak
Cypress	instalator	nie jest konieczne	prosty
Katalon	instalator	nie jest konieczne	zaawansowany
Ranorex	instalator	nie jest konieczne	zaawansowany
Robot Framework	instalacja z poziomu konsoli lub instalacja z kodu źródłowego	interpreter	brak
Selenium	dołączenie biblioteki	kompilator wybranego języka	brak
TestComplete	instalator	nie jest konieczne	zaawansowany
UFT	instalator	przydatne ale niekonieczne	zaawansowany

Tabela 10: Charakterystyka poszczególnych narzędzi.

Nazwa	Możliwość personalizacji	Wymagane umiejętności	Sposób tworzenia testów
Cucumber	wysoka	dobra znajomość Java, C#, Ruby, JavaScript, R lub Python	skrypt
Cypress	średnia	podstawowa znajomość programowania	skrypt
Katalon	wysoka	pomocna znajomość Groovy	skrypt, wykonanie nagrywania testu, ręczne wybieranie
Ranorex	wysoka	pomocna znajomość C#	wykonanie nagrywania testu, ręczne wybieranie
Robot Framework	wysoka	znajomość podstaw języka Python	skrypt
Selenium	wysoka	dobra znajomość Java, C#, Ruby, JavaScript, R lub Python	skrypt
TestComplete	wysoka	pomocna znajomość JavaScript, JScript, Python, VBScript, DelphiScript, C#Script, C++Script	wykonanie nagrywania testu, ręczne wybieranie
UFT	wysoka	pomocna znajomość VBS	wykonanie nagrywania testu manualnego lub skrypt

Każde z rozwiązań oceniono w zależności od sposobu instalacji, wymaganego dodatkowego oprogramowania, dostępności i zaawansowania interfejsu, możliwości personalizacji narzędzia, wymaganej wiedzy przed rozpoczęciem tworzenia testów oraz sposobu tworzenia testów. Charakterystykę poszczególnych narzędzi przedstawiono w tabeli 9. oraz 10.

Tabela 11. zawiera ocenę w skali 1-3 odpowiednio od najłatwiejszego do najtrudniejszego rozwiązania w każdej z kategorii. Najłatwiejsze rozwiązanie to takie, które uzyskało najmniejszą sumę punktów. Następującym numerom w tabeli 11 przyporządkowano kategorie: 1- Instalacja, 2- Dodatkowe oprogramowanie, 3- Interfejs, 4- Możliwość personalizacji, 5- Wymagane umiejętności, 6- Sposób tworzenia testów.

Tabela 11: Trudności konfiguracji oraz obsługi narzędzia.

Nazwa	Numer kategorii					
	1	2	3	4	5	6
Cucumber	3	2	3	1	3	3
Cypress	1	1	1	2	2	3
Katalon	1	1	2	1	1	1
Ranorex	1	1	2	1	1	1
Robot Framework	2	2	3	1	2	1
Selenium	3	2	3	1	3	3
TestComplete	1	1	2	1	1	1
UFT	1	2	2	1	1	1

7.3. Szybkość wykonania testu

Aby określić który z wybranych frameworków najlepiej wypadł w testach biorąc pod uwagę szybkość wykonywania testu postanowiono nadać odpowiednio punkty w zależności od zajętego miejsca w tabelach od 5 do 9, przy czym najniższa liczba punktów ustanawia narzędzie najlepszym.

Tabela 12: Ocena narzędzi w zależności od szybkości wykonania testu.

Nazwa	Pozycja w tabelach dotyczących szybkości o przypadkach nr					Suma
	1.1	2.1	3.1	4.1	4.1-Awaria	
Selenium	4	4	1	4	1	14
Cucumber	5	1	6	6	3	21
Robot	8	2	7	7	5	29
Cypress	6	8	3	1	2	20
UFT	2	5	4	2	6	19
Ranorex	3	6	5	3	8	25
TestComplete	1	3	2	5	4	15
Katalon	7	7	8	8	7	37

W tabeli 12. zebrano punkty jakie otrzymały narzędzia analizowane w niniejszej pracy.

Najlepszym narzędziem do testowania automatycznego pod względem szybkości okazało się Selenium, bardzo zbliżony wynik, bo różniący się tylko o jeden punkt uzyskał program TestComplete. Najwolniej testy wykonywało narzędzie Katalon Studio. Pozostałe narzędzia zebrały podobną liczbę punktów.

7.4. Prezentacja wyników

W celu analizy sporządzonych testów, większość z narzędzi tworzy raporty zawierające między innymi czas trwania testu i informacje o powodzeniu próby. Tabela 13. przedstawia zbiór informacji dotyczących raportowania w wykorzystywanych narzędziach.

Tabela 13: Ocena narzędzi w zależności od sposobu prezentacji wyników.

Nazwa	Typ plików generowanych po zakończeniu testu	Czy automatycznie generowany jest raport?
Cucumber	json, html, JUnit	nie
Cypress	raport jest dołączany do projektu, brak możliwości otworzenia go bez użycia programu, możliwość eks-portu do json, html	tak
Katalon	raport jest dołączany do projektu, brak możliwości otworzenia go bez użycia programu, możliwość eks-portu do PDF, CSV, html lub JUnit	tak
Ranorex	rxlog- specjalny plik generowany przez program Ranorex (brak możliwości otworzenie pliku bez posiadania aplikacji Ranorex)	tak
Robot Framewrok	html, xml, jpg (zrzuty ekranu w wypadku wyniku negatywnego)	tak
Selenium	narzędzie samo w sobie nie generuje plików	nie
TestComplete	raport jest dołączany do projektu, brak możliwości otworzenia go bez użycia programu, możliwość eksportu do JUnit reports, MHT, HTML, XML, PDF, tcLogX	tak
UFT	raport jest dołączany do projektu, brak możliwości otworzenia go bez użycia programu. Możliwość eksportu do html	tak

Aby ocenić i wybrać najlepsze narzędzie postanowiono przyznać odpowiednio przyznać punkty. Sposób oceniania:

- zero punktów za brak generowania raportu,
- dwa punkty w przypadku generowania jednego typu pliku zawierającego raport,
- przyznanie trzech punktów jeśli generowane są minimum dwa typy plików zawierających raport,
- odjęcie jednego punktu za brak możliwości otworzenia raportu bez użycia narzędzia,

dodatkowo:

- przyznanie trzech punktów za automatyczne generowanie raportu.

Po zsumowaniu punktów otrzymujemy następujące zestawienie przedstawione w tabeli 14. W tym przypadku im więcej punktów zdobyło narzędzie tym okazuje się lepszym rozwiązaniem. Najlepszym narzędziem w tym wypadku okazał się Robot Framework

Tabela 14: Ocena podsumowująca narzędzi w zależności od sposobu prezentacji wyników.

Nazwa	Suma punktów
Cucumber	4
Cypress	6
Katalon	6
Ranorex	4
Robot Framework	7
Selenium	0
TestComplete	6
UFT	4

7.5. Podsumowanie

Wybrane rozwiązania przetestowano oraz przeanalizowano pod różnym kątem. W podrozdziałach 7.1-7.4 zostały przedstawione różne aspekty dotyczące efektywności oraz oceniono każde z narzędzi. W niniejszym podrozdziale postanowiono nadać punkty na podstawie miejsca jakie zdobyły w wybranych kategoriach w celu przekonania się o tym, które z narzędzi jest najlepsze (tabela 15.). Numery porządkowe w tabeli 15 odpowiadają kolejno: 1- dokumentacja, 2- konfiguracja, 3- szybkość, 4- raporty.

Tabela 15: Podsumowanie wszystkich analizowanych aspektów.

Nazwa	Numer porządkowy				Suma
	1	2	3	4	
Cucumber	2	5	5	3	15
Cypress	3	3	4	2	12
Katalon	4	1	8	2	15
Ranorex	7	1	6	3	17
Robot Framework	4	4	7	1	16
Selenium	1	5	1	4	11
TestComplete	5	1	2	2	10
UFT	6	2	3	3	14

8. Wnioski

Celem pracy było porównanie wybranych rozwiązań wykorzystywanych w testowaniu automatycznym aplikacji internetowych. Przeanalizowano ogólnodostępną literaturę w tym artykuły udostępnione w czasopismach naukowych. Dostarczone informacje nie pozwoliły jednak na jednoznaczne określenie, który z programów do testowania jest najlepszym rozwiązaniem.

Podsumowując badania przedstawione w tabelach, najlepszym narzędziem biorąc pod uwagę czas wykonania testu okazało się Selenium. Zbliżony wynik uzyskał program TestComplete, jako trzecie narzędzie uzyskujące najlepszą liczbę punktów okazało się UFT. Programy TestComplete oraz UFT są to aplikacje płatne, o zamkniętym oprogramowaniu w przeciwieństwie do Selenium. Dzięki temu można stwierdzić, że nie tylko płatne rozwiązania są dobrym wyborem.

Kolejnym krokiem przybliżającym do uzyskania celu niniejszej pracy było porównanie wybranych narzędzi pod względem dostępności dokumentacji oraz wsparcia społeczności, poziomu trudności konfiguracji środowisk, generowanych raportów z testów oraz szybkości wykonywania testów. W każdym z wymienionych aspektów oceniono narzędzia, nadając im subiektywne noty, oraz obiektywne w przypadku oceniania szybkości wykonania testu. Jako najlepsze narzędzie uznano program TestComplete, drugim w kolejności Selenium zaś trzecim program Cypress. Płatne narzędzie okazało się być najwygodniejszym i najłatwiejszym w obsłudze, jednak rozwiązania darmowe również nie odbiegają znacząco od rozwiązań z kosztowną licencją.

Podczas wyboru narzędzia do testowania automatycznego aplikacji internetowych należy zastanowić się jakie funkcjonalności są najbardziej pożądane. Nie należy kierować się zawsze ścisłymi liczbami oraz rankingami, ponieważ istnieje prawdopodobieństwo narażenia się na zakup narzędzia, które nie jest potrzebne. Warto wziąć również pod uwagę aspekt ludzki, czy osoba używająca narzędzia będzie potrafiła go w pełni obsłużyć.

Literatura

[1] R. Dahiya, A. Shahid, Importance of manual and automation testing 2019.

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

[3] Automatyzacja testów musi być przemyślana, https://www.computerworld.pl/news/Automatyzacja-testow-musi-byc-przemyslana,381609.html, [17.03.2020].

[4] Stowarzyszenie Jakości Systemów Informatycznych, Certyfikowany tester, Sylabus poziomu podstawowego certyfikatu ISTQB 2018, Wersja 1.0.1. 2019.

[5] H. V. Gamido, M. V. Gamido, Comparative review of the features of automated software testing tools, International Journal of Electrical and Computer Engineering (IJECE) 9(5) (2019) 4473.

[6] A. S. Gadwal, L. Prasad, Comparative review of the literature of automated testing tools 2020.

[7] D. Raghuvanshi, Introduction to Software Testing, International Journal of Trend in Scientific Research and Development (IJTSRD) 2020.

[8] D. Chelimsky, D. Astels, Z. Dennis, A. Hellesøy, B. Helmkamp, D. North, The RSpec Book: Behaviour-Driven Development with RSpec, Cucumber and Friends, The Pragmatic Bookshelf 2012.

[9] 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.

[10] A. D. Wac, T. K. Watras, G. Kozieł, Analiza porównawcza rozwiązań wykorzystywanych w testowaniu automatycznym. Journal of Computer Sciences Institute 15 (2020) 156-163.

[11] H. Q. Jaleel, Testing Web Applications, SSRG International Journal of Computer Science and Engineering 6(12) (2019) 1-9.

[12] Koszty i wartość automatyzacji. https://testerzy.pl/baza-wiedzy/koszty-i-wartosc-automatyzacji, [20.06.2020].

[13] P. Marek, Weryfikacja i automatyzacja procesu testowania oprogramowania. C0RE Magazine 2011.

[14] 10 najlepszych narzędzi automatyzacji testów. https://testerzy.pl/narzedzia/10-najlepszych-narzedzi-automatyzacji-testow, [20.06.2020].

[15] Top 10 Automation Testing Tools in 2020. https://www.netsolutions.com/insights/top-10-automation-testing-tools/, [20.06.2020].