JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 29 (2023) 374–382
Received: 30 June 2023
Accepted: 29 August 2023

Comparison of tools for creating and conducting automated tests

Porównanie narzędzi do tworzenia i przeprowadzania testów automatycznych

Grzegorz Wojciech Bielesza*, Mariusz Dzieńkowski

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

Abstract
The aim of the article was to compare three tools for preparing and conducting automated tests - JUnit5, TestNG and Spock. The study involved the execution of a series of three experiments that consisted of performance tests based on five test scenarios, functional tests checking the capabilities of each tool, and popularity and support tests. The performance experiment consisted in setting up the test environment, preparing tests based on developed scenarios, executing them multiple times, and then verifying and analysing the obtained time results. The comparison of tools capabilities was based on the extracting of a set of key functionalities of the software used to develop tests. The last analysis was based on a comparison of the popularity of the tested platforms, and was performed on the basis of three popular websites providing information on user activity within the test tool.

Keywords: automated testing; JUnit5; TestNG; Spock

Streszczenie
Celem artykułu było porównanie trzech narzędzi do przygotowywania i przeprowadzania testów automatycznych – JUnit5, TestNG oraz Spock. Zrealizowane badania obejmowały wykonanie serii trzech eksperymentów, na które się składały testy wydajnościowe, oparte na pięciu scenariuszach testowych, testy funkcjonalne sprawdzające możliwości każdego z narzędzi oraz testy popularności i wsparcia. Eksperyment wydajnościowy polegał na skonfigurowaniu środowiska testowego, przygotowaniu testów bazujących na opracowanych scenariuszach, wielokrotnym ich wykonaniu, a następnie weryfikacji i analizie uzyskanych wyników czasowych. Porównanie możliwości narzędzi opierało się na wyodrębnieniu zestawu kluczowych funkcjonalności oprogramowania służącego do opracowywania testów. Ostatnia analiza polegała na porównaniu popularności badanych platform testowych i zrealizowana została na podstawie trzech popularnych serwisów internetowych udostępniających informacje o aktywności użytkowników w obrębie danego narzędzia testowego.

Słowa kluczowe: testowanie automatyczne; JUnit5; TestNG; Spock

*Corresponding author
Email address: grzegorz.bielesza@pollub.edu.pl (G. W. Bielesza)

1. Wstęp

Występowanie błędów w systemach informatycznych spowodowanych niepoprawną implementacją może wiązać się z różnego rodzaju następstwami. Wykrycie niewielkiej usterki w środowisku deweloperskim, może skutkować szybką eliminacją nieprawidłowości, nieniosącą za sobą żadnych odczuwalnych, negatywnych konsekwencji dla użytkowników. Istnieją jednak przypadki, w których wystąpienie luk w oprogramowaniu w środowisku produkcyjnym spowodowało ograniczenie dostępności aplikacji, a niektóre problemy z bezpieczeństwem czy też stabilnością systemu były przyczyną poważnych kłopotów finansowych czy prawnych, a nawet doprowadzały do wycofania systemu z rynku.

Testy to niezbędny element każdego procesu wytwarzania oprogramowania. Bez odpowiedniego przetestowania napisanego kodu, praktycznie nieosiągalne jest poprawne działanie programu. Dotyczy to zarówno małych, jak i dużych systemów. Im większy system, tym większy potencjał na występowanie luk w kodzie, które docelowo mogą prowadzić do poważnych problemów podczas fazy utrzymania oprogramowania.

Wobec powyższego, większość profesjonalnych zespołów produkujących oprogramowanie stawia duży nacisk na testowanie wprowadzanych zmian w kodzie. Bywają przypadki, w których testami zajmują się deweloperzy oraz takie, w których odpowiedzialność za odpowiednie przetestowanie funkcjonalności spoczywa na wykwalifikowanych w tym aspekcie testerach.

Wykrycie wszystkich błędów w oprogramowaniu jest niewykonalne, zwłaszcza gdy badany jest duży system, jednakże odpowiednio przeprowadzone testy programu, zdecydowanie zmniejszają ryzyko wystąpienia nieprzewidzianych skutków ubocznych związanych z implementacją, jednocześnie pozwalając na doprowadzenie jej do satysfakcjonującego poziomu poprawności, szczególnie dla kluczowych funkcjonalności systemu.

Na rynku istnieje wiele narzędzi do przygotowania i realizacji różnego typu automatycznych testów wykonywanych na oprogramowaniu. Istotnym problemem jest wybór optymalnego narzędzia dostosowanego do potrzeb. W związku z tym, celem niniejszego artykułu jest analiza porównawcza trzech popularnych platform do tworzenia i przeprowadzania testów automatycz-

nych, oparta na trzech kryteriach: funkcjonalności, wydajności oraz perspektyw rozwoju oprogramowania. W badaniach zostały uwzględnione testy jednostkowe, integracyjne oraz testy typu end-to-end. Porównywane były narzędzia umożliwiające testowanie aplikacji napisanych w języku Java.

W oparciu o powyższy cel określono następujące hipotezy badawcze:
H1: JUnit5 może umożliwić uzyskanie najlepszego czasu dla testów jednostkowych o niskim poziomie skomplikowania funkcjonalnego.
H2: Zastosowanie parametryzacji testów w przypadku dużych zbiorów danych może poskutkować najlepszym wynikiem wydajnościowym dla narzędzia TestNG w stosunku do wyników uzyskanych przez pozostałe badane narzędzia.
H3: Najpopularniejszym oprogramowaniem do przygotowywania i przeprowadzania testów spośród badanych narzędzi jest JUnit5.

2. Przegląd literatury

Wdrożenie oprogramowania powinno wiązać się z dostarczeniem sprawdzonego produktu, który przechodząc w fazę utrzymania powinien cechować się stabilnością i wysoką niezawodnością. W celu spełnienia tych dwóch kryteriów rozwijane systemy są wieloaspektowo testowane. Wykorzystywane są różne techniki, narzędzia i rodzaje testów, a mnogość rozwiązań jest częstym powodem dyskusji występującym w wielu publikacjach zajmujących się tą tematyką. Przykładem pracy, poruszającej te zagadnienia jest artykuł pt. „Towards Automated Software Testing: Techniques, Classifications and Frameworks", w której autor przedstawia różne rozwiązania dla testów automatycznych [1].

Rozwijanie systemu można oprzeć na wielu różnych technikach, w których testy odgrywają istotną rolę. Publikacja pt. „Combination of test-driven development and behavior-driven development for improving backend testing performance" porusza tematy metodyk Test-Driven-Development oraz Behaviour-Driven-Development. Badania przeprowadzone przez autorów artykułu skupiają się na analizie połączenia obu tych technik w tak zwane T-BDD. Badacze z powodzeniem pokazują, że kombinacja obu tych metodyk pozwala na osiągnięcie wysokiego procentu pokrycia testami oraz elastyczności testów na zmiany parametrów metod [2].

Istnieje wiele klasyfikacji testów takich jak chociażby podział na testy manualne polegające na ręcznym sprawdzaniu poprawności zaimplementowanych funkcjonalności oraz testy automatyczne wykonywane przez specjalistyczne oprogramowanie. W pracy „Web based Automation Testing and Tools", autor w jednym z rozdziałów dokonuje porównania testów manualnych i automatycznych. Analizy przeprowadza na podstawie badań szybkości wykonania się testów, ich precyzji oraz opłacalności. Dodatkowo publikacja ta zawiera zestawienia innych rodzajów testów. Porównywane są testy białej i czarnej skrzynki oraz testy statyczne i dynamiczne [3].

Problematyka testowania automatycznego poruszana jest w bardzo wielu dyskusjach naukowych. Przykładem pracy dotyczącej tej tematyki jest artykuł pt. „A Study of Automated Software Testing: Automation Tools and Frameworks", w której autor zwraca uwagę na przewagę testów automatycznych w aspekcie szybkiego dostarczania wysokiej jakości oprogramowania [4].

Odnosząc się ponownie do kwestii testów automatycznych, istnieje inna klasyfikacja, dzieląca testy w zależności od obszaru i sposobu testowania. W związku z tym można wyróżnić izolowane testy jednostkowe, testy integracyjne sprawdzające zależności pomiędzy oddzielnymi komponentami aplikacji oraz testy end-to-end realizujące scenariusze testowe. Temat testów integracyjnych został poruszony w ramach pracy pt. „Integration testing of object-oriented software", która dotyczy tematu testowania systemów obiektowych ze szczególnym uwzględnieniem zagadnień związanych z testami integracyjnymi. W pracy tej zaprezentowano różne poziomy testowania integracji oraz przedstawiono sposoby adaptacji tradycyjnych strategii integracyjnych [5].

Współcześnie, testowanie oprogramowania najczęściej wiąże się z korzystaniem z zaawansowanych narzędzi lub szkieletów usprawniających pracę. W artykule o nazwie „Analiza i porównanie szkieletów programistycznych używanych do testów automatycznych" została przeprowadzona analiza porównawcza zestawiająca narzędzia JUnit oraz TestNG, która polegała na porównaniu składni obu frameworków oraz testach wydajnościowych. W wyniku analizy stwierdzono, że czasy uruchomienia poszczególnych przypadków testowych są podobne [6].

W przypadku testowania kodu napisanego w języku Java, najpopularniejszym narzędziem do przeprowadzania testów jest JUnit – narzędzie, które przeanalizowano w pracy pod tytułem „Automated Java Testing: JUnit versus AspectJ". W artykule tym autorzy porównują dwa popularne narzędzia do testowania automatycznego JUnit i AspectJ. Podczas analizy, badający dokonali zestawienia podobieństw obu systemów, a następnie szczegółowo porównali je pod kątem takich kryteriów jak liczba linii kodu testowego czy krzywa uczenia się. Ostatecznym wynikiem analizy było stwierdzenie, że w przypadku programowania zorientowanego aspektowo (AOP), AspectJ jest skuteczniejszy niż JUnit [7].

Kolejna analiza dotyczyła porównania frameworków TestNG oraz WebdriverIO. Została ona opisana w artykule pt. „Analiza porównawcza frameworków do automatyzacji testowania aplikacji webowych na przykładzie TestNG i WebdriverIO". W ramach tej pracy określono kryteria porównawcze odnoszące się do dwóch kwestii: przebiegu tworzenia testów oraz ich wydajności. Na podstawie uzyskanych wyników z badań wydajność w przeprowadzaniu testów narzędziem WebdriverIO okazała się być wyższa niż wydajność przeprowadzania testów we frameworku TestNG. Jeśli chodzi o kryteria ogólne, WebdriverIO okazał się być łatwiejszy we wdrożeniu [8].

W pracy „Comparative Study of Automated Testing Tools: Selenium, Quick Test Professional and Testcomplete" zostały przedstawione i porównane trzy frameworki wykorzystywane do tworzenia i uruchamiania testów automatycznych. Zestawione narzędzia to Selenium, Quick Test Professional oraz TestComplet. Analiza porównawcza wykonana w ramach artykułu opierała się na kryterium funkcjonalności. Według autorów publikacji, najlepszą platformą testową okazał się Quick Test Professional będąc narzędziem wszechstronnym i przystosowanym dla krytycznych i bardziej ryzykownych aplikacji [9].

3. Metody badawcze

Badania przeprowadzone zostały w oparciu o aplikację testową napisaną w języku Java. Zbiór narzędzi, które wykorzystano w badaniach obejmuje:

- szkielet programistyczny Spring Boot;
- środowisko programistyczne Intellij IDEA;
- narzędzia testowe: JUnit5, Spock, TestNG.

Środowisko testowe służące do pomiarów szybkości wykonywania testów zostało zaprezentowane w Tabeli 1.

Tabela 1: Parametry środowiska testowego

Procesor	AMD Ryzen 5 2600
Płyta główna	Gigabyte B450m DS3H
Pamięć RAM	Patriot 16GB 3200MHz CL16 Viper 4
Dysk	ADATA 512GB M.2 PCIe NVMe XPG GAMMIX S11 Pro
Karta graficzna	ASUS Radeon RX 580 Dual OC 4GB GDDR5

Aplikacja użyta na potrzeby eksperymentu posłużyła jako środowisko bazowe, dostarczające metod umożliwiających porównanie narzędzi testowych pod względem wydajności przeprowadzania testów. Każde z narzędzi zostało wykorzystane do przeprowadzenia testów na tych samych testowalnych metodach realizujących proste algorytmy oraz korzystające z dużych zasobów komputera. Takie podejście miało na celu sprawdzenie zarówno prędkości uruchamiania testu, jak i przeprowadzenia bardziej skomplikowanych operacji, przy wykorzystaniu mechanizmów dostarczanych przez dane narzędzie. Zestawienie narzędzi zostało przedstawione w postaci tabelarycznej oraz opisowej wyjaśniającej otrzymane wyniki. Testy zostały napisane przy użyciu analogicznych mechanizmów, w celu analizy wydajności poszczególnych funkcjonalności narzędzi. Każdy scenariusz został powtórzony pięciokrotnie w celu wyeliminowania błędu pomiarowego, związanego z buforem lub chwilowym opóźnieniem systemu. Na podstawie uzyskanych wyników z pięciu prób zostały obliczone: średnia, mediana i odchylenie standardowe.

Scenariusz badawczy składał się z następujących kroków:

1. instalacja oraz konfiguracja frameworka w aplikacji testowej;
2. stworzenie zbioru przykładowych testów w oparciu o dokumentację techniczną badanego narzędzia;
3. przeprowadzenie testów automatycznych z użyciem wybranego środowiska testowego;
4. zapisanie wyników czasowych wszystkich wykonanych prób;
5. analiza otrzymanych wyników: ocena skuteczności oraz szybkości na tle innych rozwiązań;
6. podsumowanie i wyciągnięcie wniosków.

Analiza funkcjonalna została zrealizowana w formie zestawienia możliwości narzędzi testowych i porównania ich, podczas którego zwrócono uwagę na wpływ oferowanych funkcjonalności na produktywność podczas procesu testowania kodu. Dodatkowo porównana została składnia oraz czytelność kodu testowego, a także ergonomia przygotowywania testów. Porównanie funkcjonalności zostało zrealizowane w odniesieniu do przygotowywania kluczowych rodzajów testów. Ze względu na brak możliwości porównania pomiarowego na rzecz zestawienia funkcjonalnego, został sporządzony szczegółowy opis i argumentacja przewagi jednego narzędzia nad innym pod względem funkcjonalnym. Całość analizy narzędzia została również przedstawiona w postaci tabelarycznej prezentującej kluczowe możliwości oferowane przez oprogramowanie (Tabela 13). Sporządzono scenariusz testowy badania funkcjonalnego składający się z następujących kroków:

1. wybór opisywanej funkcjonalności;
2. zapoznanie się z oficjalną dokumentacją środowisk testowych w kwestii możliwości oferowanych w zakresie wybranej funkcjonalności;
3. utworzenie przykładowych testów z wykorzystaniem metod oferowanych przez każde z narzędzi;
4. wykonanie symulacji testu;
5. podsumowanie i analiza wsparcia dla funkcjonalności w odniesieniu do każdego środowiska.

Porównanie popularności środowisk do testowania aplikacji zostało wykonane w formacie opisowym. W odniesieniu do popularności wyciągnięto również wnioski na temat wsparcia technicznego badanego narzędzia, aktualności oraz perspektyw rozwoju. Poniżej przedstawiono kryteria porównawcze odnoszące się do wybranych trzech serwisów internetowych, na podstawie których przygotowano analizę porównawczą.

- Github – porównanie liczby zgłoszeń, gwiazdek oraz forków, dotyczących danego frameworka;
- Stack Overflow – porównanie liczby pytań i odpowiedzi w tematach odnoszących się do badanego narzędzia;
- Google Trends – zestawienie popularności w czasie dla każdego środowiska.

4. Przebieg badań

4.1. Testy wydajnościowe

Do przeprowadzenia testów wydajnościowych została wykorzystana autorska aplikacja testowa pozwalająca na instalację narzędzi testowych za pomocą oprogramowania Maven. Program w połączeniu ze środowiskiem programistycznym umożliwiał uruchamianie

testów z poziomu kodu. Ze względu na eksperyment związany z testami typu end-to-end, w aplikacji zostały umieszczone klasy testowe: ”Room”, ”Service” oraz ”Equipment”. Posiadały one konstruktory, metody modyfikujące stan obiektów oraz umożliwiające zapis do bazy danych. Kod aplikacji testowej został użyty tylko w testach end-to-end. Pozostałe testy były autonomiczne i badały obszary, w których dane narzędzie mogło wykazywać znaczne różnice w wydajności w stosunku do innych, porównywanych platform testowych. Do celów badań opracowano pięć przypadków testowych, które zostały opisane w Tabeli 2.

Tabela 2: Scenariusze testów wydajnościowych

Przypadek testowy	**Opis**
Test jednostkowy odwracający ciąg znaków	Test został utworzony w oparciu o prostą metodę umożliwiającą generowanie palindromu z podanego tekstu. Test ma na celu sprawdzenie różnic w czasach uruchomienia narzędzi testowych. Czas samego przebiegu testu można uznać za pomijalny.
Jednostkowy test jednowątkowy wykorzystujący dużą ilość zasobów	Test sprawdzający czas wykonania operacji mocno obciążającej zasoby komputera, która wykorzystuje jeden wątek. Tymi operacjami są dwa rodzaje sortowania (bąbelkowe oraz binarne), które zostały wykonane po sobie. Algorytmy te działają na dużym zbiorze liczb. Test ma na celu sprawdzenie czy wykonanie operacji mocno obciążającej zasoby komputera jest spowalniane przez framework.
Test parametryzowany	Badanie sprawdzające czas wykonania testu z dołączoną do niego pulą danych. Ma ono na celu przeanalizowanie prędkości pobierania oraz wykorzystywania danych w ramach testu. Kod wykorzystywany na potrzeby testu polega na odwróceniu znaków w tekście zawierającym 10000 wygenerowanych wyrazów, użytych jako dane wejściowe.
Test wykorzystujący wielowątkowość	Test sprawdzający czas uruchomienia wielu testów jednocześnie z wykorzystaniem wbudowanego mechanizmu uruchamiania wielowątkowego. Test ma na celu sprawdzenie, które z porównywanych narzędzi testowych posiada najlepszą wydajność podczas jednoczesnego przeprowadzania wielu testów.
Test end-to-end	Test oparty o realną infrastrukturę sprawdzający czas wykonania identycznego scenariusza testowego dla każdego porównywanego narzędzia. Test wykorzystuje JPA, przeprowadzając operacje na bazie danych MySQL. W ramach testu tworzone są obiekty, a następnie zmieniany jest ich stan za pomocą metod, klas, których obiekty są reprezentacją. Ostatecznie obiekty zapisywane są do bazy danych.

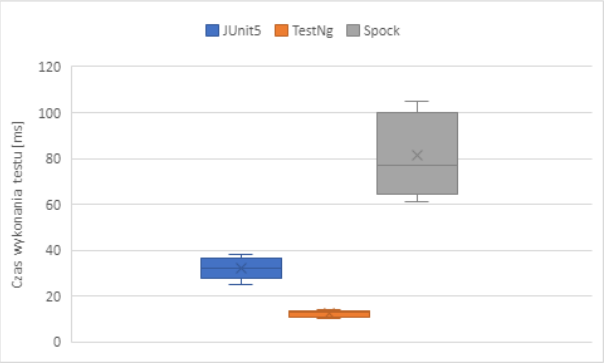
Pięciokrotny pomiar czasu wykonania testu dla każdego narzędzia, zwiększył wiarygodność przeprowadzonego badania i pozwolił na obliczenie średniej arytmetycznej, mediany oraz odchylenia standardowego. Dla każdego z przypadków testowych został przygotowany wykres pudełkowy przedstawiający rozrzut czasów wykonania, a także opis słowny otrzymanych wyników (Rysunki 1-5).

Tabela 3: Pomiary czasów dla testu odwracającego ciąg znaków

	T1 [ms]	T2 [ms]	T3 [ms]	T4 [ms]	T5 [ms]
JUnit5	25	32	35	38	30
TestNG	13	10	12	13	14
Spock	77	95	105	68	61

Tabela 4: Analiza statystyczna wyników testu odwracającego ciąg znaków

	Średnia [ms]	Mediana [ms]	SD [ms]
JUnit5	32	32	4
TestNG	12	13	1
Spock	81	77	15



Rysunek 1: Wykres pudełkowy czasów wykonania pierwszego przypadku testowego.

W pierwszym przypadku testowym, w którym porównywano czasy uruchomienia narzędzia testowego najlepiej wypadło narzędzie TestNG, a najgorzej Spock. Z wyników zawartych w Tabelach 3 i 4 oraz na Rysunku 1 można wywnioskować, że pomiędzy czasem uruchomienia JUnit5, a TestNG jest niewielka różnica, natomiast Spock uruchamia się znacznie wolniej. Średnie czasy uruchamiania narzędzia TestNG są trzy razy krótsze niż narzędzia Spock. Dla pojedynczego testu, różnica wyrażona w milisekundach może być nieodczuwalna, natomiast przy realizacji kilku tysięcy testów utworzonych podczas rozwijania dużego projektu programistycznego, różnice mogą być znaczące.

Wyniki uzyskane po zrealizowaniu drugiego scenariusza testowego (Tabela 5 i 6, Rysunek 2), w którym wykonywano operacje jednowątkowe wymagające dużej ilości zasobów, okazały się do siebie zbliżone. Różnice czasowe były w granicach błędu pomiarowego.

Oznacza to, że żadne z analizowanych narzędzi nie spowalnia wykonywania czystego kodu napisanego w języku programowania.

Tabela 5: Pomiary czasów dla testu jednowątkowego

	T1 [ms]	T2 [ms]	T3 [ms]	T4 [ms]	T5 [ms]
JUnit5	65	63	62	63	63
TestNG	65	65	64	63	63
Spock	63	61	59	62	62

Tabela 6: Analiza statystyczna wyników testu jednowątkowego

	Średnia [ms]	Mediana [ms]	SD [ms]
JUnit5	63	63	1
TestNG	64	64	1
Spock	61	62	1

Rysunek 2: Wykres pudełkowy czasów wykonania drugiego przypadku testowego.

Tabela 7: Pomiar czasów wykonania testu wykorzystującego wielowątkowość

	T1 [s]	T2 [s]	T3 [s]	T4 [s]	T5 [s]
JUnit5	19,136	20,543	15,577	15,554	18,237
TestNG	26,274	23,681	17,670	17,811	19,268
Spock	21,451	14,935	18,483	14,453	11,371

Tabela 8: Analiza statystyczna wyników testu wykorzystującego wielowątkowość

	Średnia [s]	Mediana [s]	SD [s]
JUnit5	17,809	16,907	1,711
TestNG	20,941	19,268	2,967
Spock	16,139	14,935	2,976

Rysunek 3: Wykres pudełkowy czasów wykonania trzeciego przypadku testowego.

Do osiągnięcia wielowątkowości dla JUnit5 skorzystano z pliku konfiguracyjnego umożliwiającego ustawienie trybu wielowątkowego oraz adnotacji "@Execution(ExecutionMode.CONCURRENT)". Także dla narzędzia Spock przygotowano konfigurację umożliwiającą włączenie wielowątkowości. Natomiast dla oprogramowania TestNg wykorzystano parametry "threadPoolSize" oraz "invocationCount" dla adnotacji @Test.

W przypadku badania czasów wykonania podczas uruchomienia wielu testów jednocześnie okazało się, że TestNG osiąga najgorsze wyniki. Średni czas przeprowadzenia testów dla narzędzia Spock jest około 23% krótszy względem TestNG oraz około 15% krótszy od JUnit5 (Tabele 7 i 8, Rysunek 3). Pomimo automatycznego zarządzania wątkami, ze względu na dużą liczbę zmiennych podczas przeprowadzenia tego badania widoczny jest duży rozrzut wyników.

Tabela 9: Wyniki testu parametryzowanego

	T1 [ms]	T2 [ms]	T3 [ms]	T4 [ms]	T5 [ms]
JUnit5	693	639	709	585	689
TestNG	729	670	732	792	670
Spock	324	350	313	345	314

Tabela 10: Analiza statystyczna wyników testu parametryzowanego

	Średnia [ms]	Mediana [ms]	SD [ms]
JUnit5	666	664	39
TestNG	719	729	39
Spock	329	324	13

Rysunek 4: Wykres pudełkowy czasów wykonania czwartego przypadku testowego.

W odniesieniu do testów parametryzowanych wykorzystujących kod odwracający ciąg znaków najlepiej wypadła platforma Spock, osiągając znacznie krótszy średni czas niż pozostałe dwie platformy testowe (Tabele 9 i 10, Rysunek 4). Oznacza to, że wczytywanie danych testowych oparte na parametryzacji testów jest najbardziej wydajne dla tego właśnie frameworka. Najgorzej pod względem czasowym wypadł TestNG, który jest średnio o około połowę wolniejszy od szkieletu Spock oraz nieznacznie wolniejszy od JUnit5. W przypadku JUnit5 użyto adnotacji "@ParametrizedTest" w połączeniu z "@MethodSource", dla narzędzia Spock wykorzystano adnotację "@Unroll", natomiast w dla

TestNG skorzystano charakterystycznej dla tego frameworka adnotacji "@DataProvider".

Czasy wykonania testów typu end-to-end przeprowadzonych na jednym z modułów aplikacji testowej były najkrótsze dla platform TestNG oraz JUnit5. Oba narzędzia osiągnęły wyniki na podobnym poziomie (Tabele 11 i 12, rysunek 5) i w stosunku do platformy Spock czasy były o około 120 ms krótsze. Może to wynikać z dłuższego uruchamiania się środowiska testowego i nieznacznie wolniejszego przetwarzania operacji opartych na języku Java.

Tabela 11: Wyniki testu end-to-end

	T1 [ms]	T2 [ms]	T3 [ms]	T4 [ms]	T5 [ms]
JUnit5	344	361	380	385	353
TestNG	338	391	350	342	356
Spock	494	492	466	484	499

Tabela 12: Analiza statystyczna wyników testu parametryzowanego

	Średnia [ms]	Mediana [ms]	SD [ms]
JUnit5	365	361	13
TestNG	355	350	16
Spock	487	492	10

Rysunek 5: Wykres pudełkowy czasów wykonania piątego przypadku testowego.

4.2. Testy funkcjonalne

Analiza możliwości oferowanych przez dany framework została zrealizowana w oparciu o wyspecyfikowane kryteria. Wyniki tej analizy przedstawia Tabela 13. Dodatkowo dokonano scharakteryzowania poszczególnych narzędzi, biorąc pod uwagę takie kwestie jak składnia, elastyczność oraz prostota sporządzania testów. Poza tym określono istotne wady i zalety każdego z narzędzi.

Tabela 13: Zestawienie funkcjonalności

Funkcjonalność	**JUnit5**	**TestNG**	**Spock**
Testy jednostkowe	Tak	Tak	Tak
Testy integracyjne	Tak	Tak	Tak
Asercje	Bogate wsparcie	Bogate wsparcie	Bogate wsparcie
Testy dynamiczne	Tak	Tak	Tak
Testy parametryzowane	Tak	Tak	Tak
Obsługa wyjątków	Tak	Tak	Tak
Testy równoległe	Adnotacja, plik konfiguracyjny	Adnotacja	Plik konfiguracyjny
Generowanie raportów wyników testów	Tak	Tak	Nie
Wsparcie dla tworzenia rozszerzeń	Tak	Tak	Tak
Możliwość integracji z narzędziami zewnętrznymi	Tak	Tak	Tak
Testy asynchroniczne	Tak	Tak	Nie
Możliwość zarządzania stanem testów	Tak	Tak	Tak
Możliwość wykonywania testów w kolejności	Tak	Tak	Tak
Sposób dostarczania danych testowych	Fabryki, metody, pliki	Fabryki, metody, pliki	Tabele, zbiory, metody
Wsparcie dla testów wykorzystujących interfejsy	Nie	Tak	Nie
Wsparcie dla testów wykorzystujących słuchaczy	Tak	Tak	Nie
Wsparcie dla testów wykorzystujących grupy	Tak	Tak	Tak
Wsparcie dla testów wykorzystujących wyrażenia regularne	Nie	Nie	Tak

Z Tabeli 12 wynika, że podstawowe funkcjonalności takie jak wsparcie dla testów jednostkowych, integracyjnych, parametryzowanych czy obsługa wyjątków są oferowane przez wszystkie narzędzia. Każde oprogramowanie posiada szeroki zbiór asercji. Pewne różnice w zakresie oferowanych możliwości widoczne są w przypadku narzędzia Spock. Oprogramowanie to nie wspiera testów opartych o interfejsy, testów asynchronicznych, a także nie pozwala na generowania raportów z wynikami testów. Natomiast jako jedyne spośród badanych narzędzi umożliwia przygotowywanie testów wykorzystujących wyrażenia regularne, za pomocą adnotacji. Spock nie posiada również wbudowanej adnotacji umożliwiającej dostarczanie danych z pliku, co zapewniają pozostałe dwa środowiska. Testy równoległe zarówno dla narzędzia Spock, jak i JUnit5 muszą być skonfigurowane w specjalnym pliku konfiguracyjnym. W przypadku JUnit5 możliwe jest określenie za pomocą adnotacji, które testy powinny zostać wykonane równolegle. Spock nie posiada takiej możliwości. Dodatkowo autorzy szkieletu opartego na języku Groovy określają testy równoległe jako funkcjonalność eksperymentalną dla tego narzędzia. TestNG oraz JUnit5 posiadają wsparcie dla testów asynchronicznych, natomiast tylko TestNG umożliwia tworzenie testów w oparciu o interfejsy. Dodatkowo TestNG, pozwala na zarządzanie stanem testów w największym zakresie

spośród analizowanych narzędzi. Pozwala on na wykonywanie operacji przed testami określonymi w grupie lub konkretnym pakiecie. Według tabeli 13 TestNG posiada największy zakres funkcjonalności. To narzędzie nie obsługuje jedynie testów wykorzystujących wyrażenia regularne (tzw. regexy).

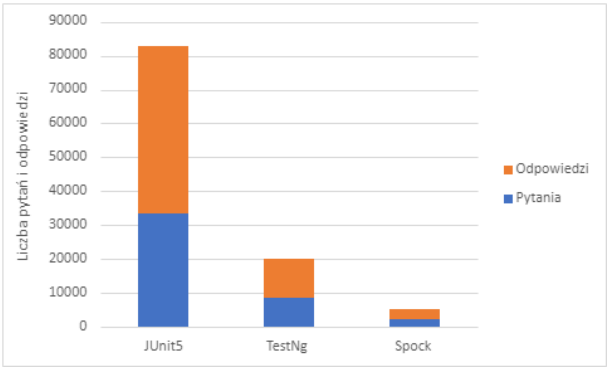
Odnośnie składni i prostoty tworzenia testów szkielety JUnit5 oraz TestNG nie posiadają ściśle zdefiniowanych ram. Komponowanie testów jest elastyczne i wymaga jedynie użycia odpowiedniej adnotacji. Przygotowywanie testów we frameworku Spock wymaga zdefiniowania klauzul given, when oraz then, a także umieszczenia, opisów obok tych klauzul zazwyczaj odnoszących się do stanu testu. Narzucenie konieczności jawnego określenia granic testów, może mieć konsekwencje w postaci podwyższenia standardu testów obecnych w aplikacji, a także może ułatwić utrzymanie kodu testowego i może być pomocne w jego zrozumieniu. Specyfika testów opracowanych w szkielecie Spock ułatwia projektowanie testów typu end-to-end ze względu na konieczność zdefiniowania poszczególnych kroków testów. W kwestii komponowania testów, wszystkie analizowane narzędzia mają podobny poziom skomplikowania.

Odnosząc się do ergonomii przygotowywania testów, JUnit5 oraz TestNG wykorzystują język Java. Ze względu na charakter języka, na którym są oparte, błędy popełnione podczas opracowywania testów są na bieżąco przetwarzane, a oprogramowanie nie pozwala na skompilowanie kodu odpowiedzialnego za realizację testu. Szkielet Spock używa języka Groovy i dlatego wyświetla błędy dopiero po zakończeniu działania testowanego kodu. Wydłuża to proces przygotowywania testów, szczególnie podczas testowania bardziej wymagających funkcjonalności aplikacji, ze względu na długi czas wykonywania kodu i wykorzystanie dużej ilość zasobów.

4.3. Testy popularności

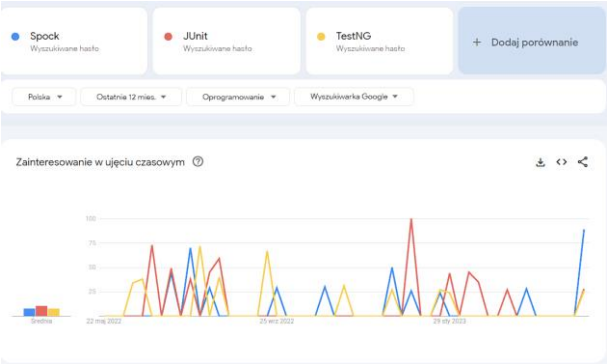
Popularność porównywanych narzędzi została przeanalizowana na podstawie aktualnych wyników zamieszczanych w trzech serwisach internetowych GitHub, StackOverflow oraz Google Trends. Analiza w serwisie GitHub, opiera się na statystykach oficjalnych projektów odpowiedzialnych za rozwój testowanego oprogramowania. Badanie na stronie StackOverflow, zostało przeprowadzone za pomocą narzędzia StackExchange umożliwiającego formułowanie zapytań SQL. Wykorzystano zapytania SQL sprawdzające liczbę zapytań i odpowiedzi w oparciu o tag reprezentujący dany framework. Analiza popularności w serwisie Google Trends została zrealizowana na podstawie liczby wyszukiwań porównywanych narzędzi w sekcji oprogramowanie oraz ich średnich wyników uwzględniających dane z ostatnich 12 miesięcy.

Według serwisu GitHub pierwsze miejsce w zestawieniu zajmuje JUnit5, osiągając wynik ponad 1300 forków, około 5700 gwiazdek oraz 110 zgłoszeń. Wybór najbardziej popularnego frameworka spośród pozostałych nie jest jednoznaczny. Kod TestNG został pobrany około 1000 razy, na jego podstawie utworzono 252 zgłoszenia, a sam framework otrzymał około 1800 gwiazdek. Szkielet Spock choć uzyskał dużo mniejszą liczbę forków, tzn. 459, to jego repozytorium zgromadziło ponad 3400 gwiazdek i 212 zgłoszeń. Te wyniki świadczą o podobnym poziomie popularności narzędzi TestNG oraz Spock w serwisie GitHub.



Rysunek 6: Wykres porównujący popularność narzędzi na platformie StackOverflow.

Przyglądając się wykorzystaniu tagów na platformie StackOverflow (Rysunek 6) można zauważyć, że sprawdzając liczbę zapytań i odpowiedzi dla tagu “junit” oraz tagu “junit5” uzyskamy zupełnie różne wyniki. Dla tagu bez podanej wersji frameworka wyniki wynoszą aż 33601 pytań oraz 49170 odpowiedzi, natomiast dla narzędzia z podaną wersją jest to zaledwie 3499 pytań oraz 4125 odpowiedzi. Spowodowane jest to tym, że tag “junit” używany jest dla wątków dotyczących wcześniejszych wersji szkieletu JUnit i taki wysoki wynik tagu “junit” ma przełożenie na wybór szkieletu JUnit5 jako najbardziej popularnego na platformie StackOverflow. Drugie miejsce zajmuje TestNG z wynikiem 8676 pytań oraz 11540 odpowiedzi, a na ostatniej pozycji plasuje się framework Spock z wynikiem 2376 pytań oraz 2900 odpowiedziami.



Rysunek 7: Wykres porównujący popularność narzędzi w aplikacji Google Trends.

Analizując wyniki przedstawione przez serwis Google Trends (Rysunek 7) jako odpowiedź na żądanie porównania liczby wyszukiwań nazw analizowanych trzech narzędzi można stwierdzić, że i w tym przypadku ponownie JUnit5 jest najpopularniejszym narzędziem, a średnia popularności dwóch pozostałych narzędzi

w okresie ostatnich 12 miesięcy była na podobnym poziomie.

Biorąc pod uwagę powyższe analizy, można stwierdzić, że spośród badanych narzędzi JUnit5 jest najpopularniejszym szkieletem programistycznym do przeprowadzania testów automatycznych. Na drugim miejscu znajduje się TestNG, natomiast ostatnie miejsce przypada frameworkowi Spock. Uwzględniając oficjalną dokumentacje techniczną, liczbę zgłoszeń wraz z odpowiedziami oraz regularne aktualizacje, można stwierdzić, że każde z wyżej wymienionych narzędzi posiada bogate wsparcie techniczne oraz ma na wysokim poziomie prowadzoną dokumentację.

5. Wnioski

Analiza porównawcza narzędzi do przeprowadzania testów automatycznych wykazała wady i zalety każdego z nich. W kwestii wydajności szkieletów do testów trudno określić, który z nich jest najlepszy. Przeprowadzone rozważania wykazały, że najszybciej uruchamia się TestNG, natomiast najwolniej Spock. Problemem TestNG okazały się testy wielowątkowe, które wypadły gorzej na tle pozostałych dwóch narzędzi, a w przypadku Spock testy end-to-end wykonywały się średnio o około 30 procent dłużej. Natomiast testy parametryzowane oparte o dostarczanie danych za pomocą mechanizmów wbudowanych we framework Spock wykonywały się w czasie o połowę krótszym niż w przypadku narzędzi JUnit5 i TestNG. Należy zwrócić uwagę, że możliwe jest uzyskanie lepszych czasów realizacji testów poprzez zastosowanie bardziej zaawansowanych technik, aczkolwiek założeniem autorów pracy były badania wykorzystujące wyłącznie podstawowe mechanizmy dostarczane przez każde badane oprogramowanie.

Porównanie funkcjonalne, pozwoliło stwierdzić, że TestNG posiada największy zasób wybranych funkcjonalności spośród testowanych narzędzi. Narzędzie to wspiera prawie wszystkie rodzaje testów wymienionych w Tabeli 13 oprócz testów opartych na wyrażeniach regularnych. Framework ten posiada również bogate wsparcie dla dostarczania danych, a także najbogatszy zasób adnotacji umożliwiających zarządzanie stanem testów. Wypada on również bardzo dobrze w przypadku ergonomii wytwarzania testów. Ze względu na wykorzystywanie języka Java, jest on bardziej praktyczny w przypadku przygotowywania dużej ilości testów lub testów odwołujących się do kodu, który do wykonania wymaga dużej ilości zasobów. Jeśli chodzi o wykorzystywaną strukturę i składnię podczas przygotowywania oraz utrzymania testów, najlepiej wypadł szkielet Spock, który narzuca sztywne ramy, które wytwarzane testy muszą spełniać, aby mogły się wykonać. Każde z porównywanych narzędzi charakteryzuje się prostotą przygotowywania testów. Uwzględniając wady oraz zalety każdego z powyższych frameworków najbardziej funkcjonalnym oprogramowaniem okazał się TestNG, który oferuje najwięcej możliwości bez dodatkowego doinstalowania (ang. out of the box).

Analizując wyniki popularności wśród trzech wybranych znanych serwisów internetowych można jednoznacznie stwierdzić, że JUnit5 jest najpopularniejszym narzędziem do przygotowywania i przeprowadzania testów. W każdym z badanych serwisów osiągnął najwyższy wynik. Przekłada się to na ilość poruszonych problemów programistycznych, co wiąże się z szybkością znajdowania rozwiązań w internecie. W konsekwencji może to przyspieszyć proces wytwarzania testów, ze względu na duże wsparcie, mnogość różnych wątków i szybszy dostęp do wiedzy na temat specyficznych zagadnień.

Wyniki uzyskane podczas przeprowadzonych badań pozwoliły na weryfikację postawionych na wstępie hipotez. Okazało się, że narzędzie JUnit5 nie jest najwydajniejszym frameworkiem do przeprowadzania prostych testów jednostkowych. Użycie TestNG może przynieść lepsze czasy wykonywania prostych testów ze względu na czas uruchomienia narzędzia. Oznacza to, że pierwsza hipoteza nie została potwierdzona. Także druga hipoteza nie została spełniona, ponieważ wyniki testów wydajnościowych pokazały, że TestNG nie jest najszybszym szkieletem wykonującym testy parametryzowane. Natomiast ostatnia hipoteza została potwierdzona - popularność frameworka JUnit5 okazała się jednoznacznie największa.

Wybór konkretnego narzędzia zazwyczaj podyktowany jest jego zastosowaniem oraz umiejętnościami osób przygotowujących testy. Wydajność może mieć znaczenie w odniesieniu do projektów programistycznych, na które składają się setki lub tysiące testów. Wówczas użycie nieoptymalnego rozwiązania może spowolnić proces wytwarzania i dalszego rozwoju oprogramowania. Dla większości systemów wybór technologii opiera się na przeanalizowaniu możliwości funkcjonalnych. W przypadku zapotrzebowania na utrzymanie określonej struktury testów najlepszym rozwiązaniem będzie Spock. Z kolei, kiedy ważniejsza będzie ergonomia przygotowywania dużej ilości testów, to ze względu na natychmiastowe wychwytywanie błędów, proces testowania może przyśpieszyć któryś z frameworków opartych na języku Java.

Literatura

[1] R. Torkar, Towards Automated Software Testing: Techniques, Classifications and Frameworks (PhD dissertation, Blekinge Institute of Technology) (2006).

[2] I. B. K. Manuaba, Combination of test-driven development and behavior-driven development for improving backend testing performance. Procedia Computer Science 157 (2019) 79-86.

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

[4] M. A. Umar, C. Zhanfang, A study of automated software testing: Automation tools and frameworks. International Journal of Computer Science Engineering (IJCSE) 6 (2019) 217-225.

[5] A. Orso, Integration testing of object-oriented software. Dottorato di Ricerca in Ingegneria Informatica e Automatica, Politecnico di Milano, 1998.

[6] D. Gromek, D. Gutek, Analysis and comparison of programming frameworks used for automated tests. Journal of Computer Sciences Institute 17 (2020) 339-344.

[7] M. Jain, D. Gopalani, Automated Java testing: JUnit versus AspectJ. International Journal of Computer and Information Engineering 11(11) (2017) 1215-1220.

[8] A. Shtokal, J. Smołka, Comparative analysis of frameworks used in automated testing on example of TestNG and WebdriverIO. Journal of Computer Sciences Institute 19 (2021) 100-106.

[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.