JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 6 (2018) 47-54
Wysłane: 2017-10-28
Przyjęte: 2017-11-02

Analiza porównawcza wybranych szkieletów służących do wstrzykiwania zależności

Rafał Szewczyk*, Małgorzata Plechawska-Wójcik*
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Artykuł przedstawia analizę porównawczą wybranych szkieletów do wstrzykiwania zależności w języku Java. Analiza polega na porównaniu szybkości działania frameworków w różnych konfiguracjach. Dla każdej technologii została stworzona osobna aplikacja, a wyniki badań zostały przedstawione w formie wykresów i tabel.

Słowa kluczowe: dependency-injection; ioc; java

*Autor do korespondencji.

Adres e-mail: rafal.szewczyk2@pollub.edu.pl, m.plechawska@pollub.pl

Comparative analysis of selected skeletons for dependency injection

Rafał Szewczyk*, Małgorzata Plechawska-Wójcik*
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. The article presents a comparative analysis of selected skeletons for dependency injection in Java. The analysis consists in comparing the speed of the frameworks in different configurations. For each technology has been created a separate application, and the results are presented in graphs and tables.

Keywords: dependency-injection; ioc; java

*Corresponding author.

E-mail address: rafal.szewczyk2@pollub.edu.pl, m.plechawska@pollub.pl

1. Wstęp

W dzisiejszych czasach wiele firm decyduje się na zmianę swoich statycznych stron internetowych na nowoczesne aplikacje. Jest to podyktowane coraz większymi oczekiwaniami klientów korzystających z takich serwisów [1]. Dodatkowo dzięki nim obsługa witryn staje się efektywniejsza [2]. Użytkownicy oczekują coraz szybszych programów, co spowodowało dynamiczny rozwój narzędzi do budowy takich projektów [3]. Decydując się na korzystanie z takiej aplikacji należy wybrać odpowiednią technologię, tak aby zapewnić ich sprawne działanie oraz wygodę korzystania dla użytkowników.

Celem artykułu jest porównanie wybranych szkieletów służących do wstrzykiwania zależności. Ich analiza została przeprowadzona ze względu na to, że wstrzykiwanie zależności jest fundamentalnym składnikiem działania takich aplikacji [4], a w sieci istnieją jedynie opisy poszczególnych technologii pomijające aspekty wydajnościowe [5][6][7]. Dodatkowo w publikacji zostanie sprawdzona teza, że aplikacje tworzone w oparciu o najbardziej rozbudowaną bibliotekę, mają znaczną przewagę nad autorskimi projektami wspierającymi wstrzykiwanie zależności.

2. Wstrzykiwanie zależności

Wstrzykiwanie zależności jest to wzorzec projektowy wykorzystywany w aplikacjach internetowych [8]. Polega na tworzeniu modułowych programów bez bezpośrednich zależności pomiędzy komponentami. Jest implementacją zasady odwróconego sterowania (IoC), która polega na tworzeniu oraz wiązaniu obiektów poza kodem aplikacji. Jest to realizowane za pomocą poszczególnych bibliotek, które dzięki wykorzystaniu specjalnego kontenera zapewniają im cały cykl życia [9]. Zastosowanie tego wzorca niesie za sobą wiele korzyści [10]. Przede wszystkim ułatwia budowę takich programów, poprawia możliwości dalszej rozbudowy oraz utrzymania. Dodatkowo jest możliwość łatwiejszego testowania poszczególnych funkcjonalności poprzez modularną strukturę oraz wyeliminowanie bezpośrednich zależności [11].

3. Przebieg badań

3.1. Środowisko testowe

Do przeprowadzenia testów wydajnościowych posłużono się laptopem Lenovo Y50-70 z procesorem Intel Core i7 4770HQ oraz 8GB pamięci RAM.

3.2. Badane technologie

Biblioteki użyte do testów :

- Spring (w wersji 5.0.0.RC3) – kompleksowe narzędzie do budowy aplikacji internetowych. Jest to jedna z najpopularniejszych technologii [6],

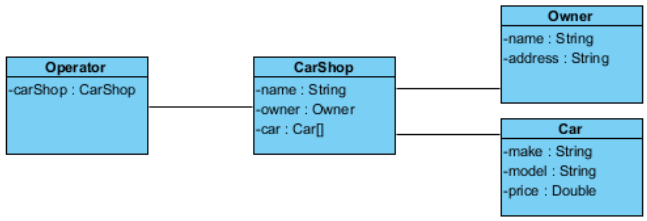
- Google Guice (w wersji 4.1.0) – framework zaprojektowany na podstawie Spring'a. Jednak skupiony na wstrzykiwaniu zależności [5],
- PicoContainer (w wersji 2.15) – jest to technologia zapewniająca wsparcie dla metodyki wstrzykiwania zależności. Jest mniej popularna od powyższych narzędzi [7],
- Weld (w wersji 3.0.1.Final) – rozbudowana technologia firmy JBoss. Posiada bogaty zestaw narzędzi do tworzenia aplikacji internetowych [12],
- EasyDI (w wersji 0.3.0) – mała biblioteka przeznaczona do projektów wykorzystujących wstrzykiwanie zależności. Ze względu na prosty kontener przeznaczona dla małych aplikacji [13],
- SilkDI (w wersji 0.6) – autorski projekt programisty Jana Bernitt'a. Stworzony w celu zwiększenia prostoty użycia oprogramowania poprzez zmniejszenie ilości konfiguracji [14].

3.3. Projekt aplikacji

Aby porównać technologie przedstawione w rozdziale 3.2, dla każdej z nich została stworzona oddzielna aplikacja. Każdy z projektów wykorzystywał 4 wzajemnie powiązane klasy:

- *Car* – klasa zawierająca informacje o samochodzie,
- *Owner* – klasa zawierająca informacje o posiadaczu samochodu,
- *CarShop* – klasa zawierająca informacje o komisie samochodowym,
- *Operator* – klasa reprezentująca posiadacza komisu.

Klasy zostały połączone relacjami. *Owner* i *CarShop* zostały połączone relacją jeden do wielu (w komisie może być wielu posiadaczy samochodów). Podobnie zostały połączone klasy *Car* i *CarShop* (w komisie może być wiele samochodów). Analogicznie zostały powiązane klasy *Operator* i *CarShop* (jeden właściciel może posiadać wiele komisów). Diagram klas takiej aplikacji został przedstawiony na rysunku 1.



Rys. 1. Diagram klas użytych w programie

3.4. Metody badawcze

Do porównania frameworków przeprowadzono badania składające się z dwóch etapów - pomiaru czasu startu kontenera oraz wstrzykiwania obiektów o różnej złożoności w ramach działania pojedynczego programu. Czas uzyskany z poszczególnych eksperymentów został przedstawiony w milisekundach, a wyniki zostały zgromadzone w tabelach oraz na wykresach.

Ponadto pomiary zostały zrealizowane dla dwóch zakresów widoczności obiektów:

- singleton - za każdym razem wstrzykiwana ta sama instancja obiektu,
- prototype – za każdym razem wstrzykiwany nowy obiekt.

Aby szczegółowo porównać zachowanie technologii podczas wstrzykiwania zależności wykorzystano obiekty o różnej strukturze:

- obiekt klasy *Car*,
- obiekt klasy *Operator* zawierający 100 obiektów *CarShop*,
- obiekt klasy *Operator* zawierający 10000 obiektów *CarShop*,
- obiekt klasy *Operator* zawierający 100000 obiektów *CarShop*.

Dodatkowo w celu uwidocznienia różnic między technologiami, podczas porównań wykorzystywanym obiektom zostały przypisane wagi według przyjętej specyfikacji:

- obiekt klasy *Car* – waga 10 %,
- obiekt klasy *Operator* zawierający 100 obiektów *CarShop* – waga 20 %,
- obiekt klasy *Operator* zawierający 10000 obiektów *CarShop* – waga 30 %,
- obiekt klasy *Operator* zawierający 100000 obiektów *CarShop* – waga 40 %.

Najwyższa pozycja w rankingu oznacza, że czas wstrzykiwania obiektów w określonej technologii był najdłuższy. Ponadto najlepsze wyniki zostały oznaczone kolorem zielonym, a najgorsze kolorem czerwonym.

Aby rezultaty otrzymane z przeprowadzonych eksperymentów były jak najbardziej zbliżone do siebie, to w wszystkich aplikacjach skorzystano z tej samej metody wstrzykiwania zależności (została użyta metoda wstrzykiwania przez pola klasy). Wyeliminowało to konieczność uruchamiania przez kontener dodatkowych metod (np. metod ustawiających).

4. Analiza wyników

4.1. Czas startu kontenera

Pierwszy etap testów składał się z pomiaru czasu, jaki niezbędny jest do uruchomienia kontenera zawierającego używane obiekty w poszczególnych technologiach.

- **Eksperyment 1**

Eksperyment pierwszy polegał na pomiarze czasu startu kontenera dla obiektów w zakresie Singleton podczas pierwszego uruchomienia programu. Rezultaty umieszczono w tabeli 1.

Tabela 1. Czas pierwszego uruchomienia kontenera, zakres singleton [ms]

	Car	Operator (100 obiektów CarShop)	Operator (1000 obiektów CarShop)	Operator (100000 obiektów CarShop)
	Czas wykonania [ms]			
Spring	449,00	447,00	460,00	**938,00**
Google Guice	353,00	203,43	205,80	202,10
PicoContainer	31,12	32,44	36,01	31,99
Weld	**661,00**	**642,00**	**631,00**	621,00
EasyDI	**3,34**	**3,53**	**3,64**	**3,24**
Silk DI	52,20	54,93	55,26	51,79

Z przeprowadzonych badań wynika, że czasy startu kontenera w każdej z technologii znacznie się od siebie różnią. Ponadto widać, że w zakresie widoczności Singleton na start kontenera nie ma wpływu ilość używanych obiektów. Aby precyzyjnie porównać technologie sporządzono ich ranking za pomocą wag opisanych w rozdziale 3.4, a rezultaty zamieszczono w tabeli 2.

Tabela 2. Ranking technologii na podstawie czasu pierwszego uruchomienia kontenera, zakres singleton

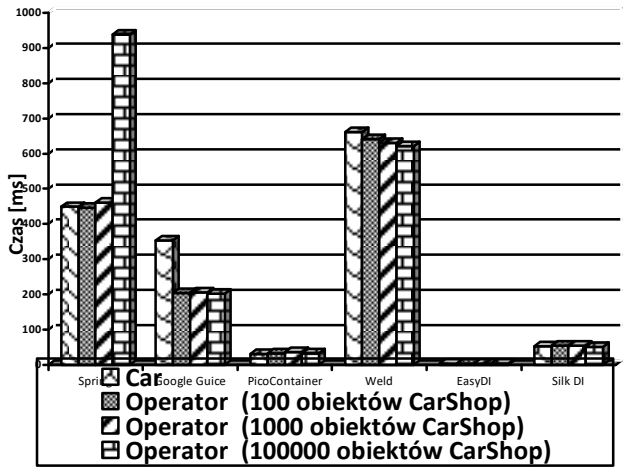
	Pozycja w rankingu
Spring	**1**
Google Guice	3
PicoContainer	5
Weld	2
EasyDI	**6**
Silk DI	4

Dzięki takiemu zestawieniu wyników widać znaczną przewagę lekkich bibliotek w tym eksperymencie. Najlepsze rezultaty osiągnęła biblioteka EasyDI, a najgorsze Weld.

Dodatkowo bazując na osiągniętych czasach można wyróżnić 2 grupy frameworków w kolejności od najgorszych do najlepszych:

1) Spring, Weld oraz Google Guice,
2) Silk DI, PicoContainer, oraz EasyDI, które są znacznie mniej rozbudowanymi technologiami.

Wyniki pierwszego eksperymentu zostały przedstawione na wykresie 1.



Wykres 1. Czas pierwszego uruchomienia kontenera, zakres singleton

- **Eksperyment 2**

Kolejny eksperyment polegał na pomiarze czasu startu kontenera dla obiektów o zakresie Singleton podczas 500 kolejnych uruchomień programu. Uśrednione wyniki zostały zawarte w tabeli 3.

Tabela 3. Średni czas uruchomienia kontenera, zakres singleton

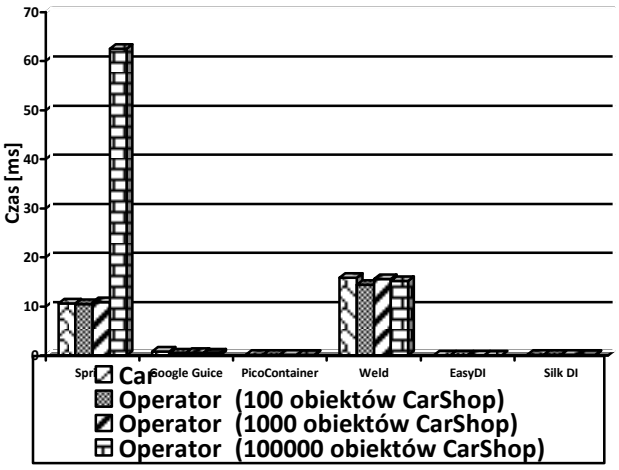
	Car	Operator (100 obiektów CarShop)	Operator (1000 obiektów CarShop)	Operator (100000 obiektów CarShop)
	Czas wykonania [ms]			
Spring	10,64	10,44	10,83	**62,59**
Google Guice	0,85	0,46	0,48	0,46
PicoContainer	0,09	0,10	0,09	0,10
Weld	**15,82**	**14,43**	**15,56**	15,24
EasyDI	**0,02**	**0,01**	**0,01**	**0,01**
Silk DI	0,18	0,19	0,18	0,18

Również w tym przypadku widać, że występują znaczne różnice między czasami. Jednak w przypadku kolejnych uruchomień programu są to zdecydowanie lepsze wyniki. Na podstawie rezultatów z tabeli 3 został stworzony ranking, który przedstawia tabela 4. Świadczy on o przewadze mniej rozbudowanych technologii.

Tabela 4. Ranking technologii na podstawie średniego czasu uruchomienia kontenera, zakres singleton

	Pozycja w rankingu
Spring	**1**
Google Guice	3
PicoContainer	5
Weld	2
EasyDI	**6**
Silk DI	4

Wyniki drugiego eksperymentu zostały przedstawione na wykresie 2.



Wykres 2. Średni czas uruchomienia kontenera, zakres singleton

- **Eksperyment 3**

Eksperyment trzeci polegał na pomiarze czasu startu kontenera dla obiektów w zakresie Prototype podczas

pierwszego uruchomienia programu. Rezultaty zostały zamieszczone w tabeli 5.

Tabela 5. Czas pierwszego uruchomienia kontenera, zakres prototype

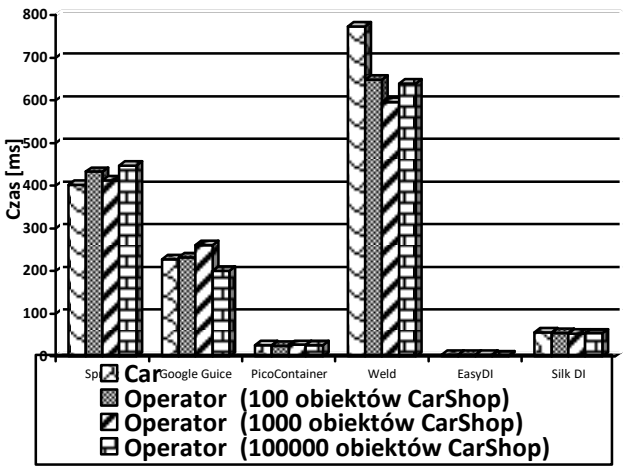
	Car	Operator (100 obiektów CarShop)	Operator (1000 obiektów CarShop)	Operator (100000 obiektów CarShop)
	Czas wykonania [ms]			
Spring	402,00	433,00	413,00	447,00
Google Guice	228,06	232,53	261,26	200,21
PicoContainer	26,49	25,40	27,26	26,45
Weld	**773,00**	**649,00**	**596,00**	**640,00**
EasyDI	**3,70**	**4,10**	**3,39**	**2,32**
Silk DI	56,87	54,78	53,71	54,22

Ponownie można zaobserwować znaczne różnice pomiędzy poszczególnymi technologiami. Dodatkowo okazało się, ze na start kontenera nie ma wpływu zakres widoczności używanych obiektów, ani ich liczba. Na podstawie rezultatów z tabeli 5 został stworzony ranking, który przedstawia tabela 6.

Tabela 6. Ranking technologii na podstawie pierwszego uruchomienia kontenera, zakres prototype

	Pozycja w rankingu
Spring	2
Google Guice	3
PicoContainer	5
Weld	**1**
EasyDI	**6**
Silk DI	4

Tu także lepsze w rankingu są małe biblioteki, które są wydajniejsze od rozbudowanych technologii. Rezultaty tego eksperymentu zostały przedstawione na wykresie 3.



Wykres 3. Czas pierwszego uruchomienia kontenera, zakres prototype

- **Eksperyment 4**

Eksperyment czwarty polegał na pomiarze czasu startu kontenera dla obiektów o zakresie Prototype podczas 500 kolejnych uruchomień programu. Wyniki osiągnięte z tego eksperymentu zostały uśrednione, a następnie umieszczone w tabeli 7.

Tabela 7. Średni czas uruchomienia kontenera, zakres prototype

	Car	Operator (100 obiektów CarShop)	Operator (1000 obiektów CarShop)	Operator (100000 obiektów CarShop)
	Czas wykonania [ms]			
Spring	**14,74**	**17,88**	**31,67**	**32,38**
Google Guice	0,50	0,46	0,44	0,46
PicoContainer	0,07	0,07	0,06	0,06
Weld	**15,40**	**21,26**	**41,56**	**51,00**
EasyDI	**0,01**	**0,01**	**0,02**	**0,02**
Silk DI	0,17	0,29	0,34	0,33

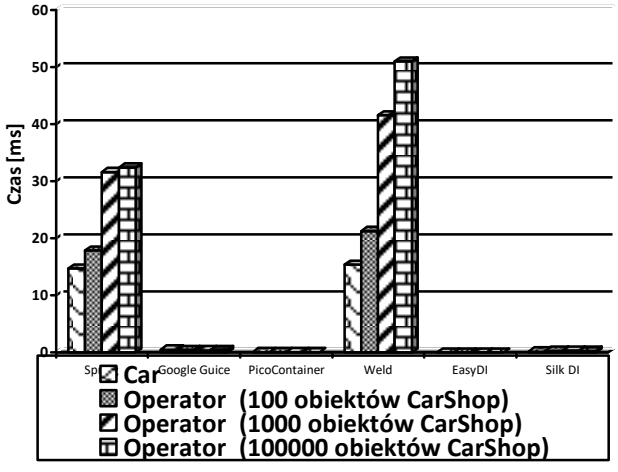
Tak samo jak w przypadku eksperymentu 2 czasy są zdecydowanie lepsze niż podczas pierwszego uruchomieniu programu. Świadczy to o konieczności załadowania obiektów do kontenera podczas pierwszej iteracji.

W tabeli 8 umieszczono ranking technologii na podstawie osiągniętych czasów.

Tabela 8. Ranking technologii na podstawie średniego czasu uruchomienia kontenera, zakres prototype

	Pozycja w rankingu
Spring	2
Google Guice	3
PicoContainer	5
Weld	**1**
EasyDI	**6**
Silk DI	4

Podobnie jak w poprzednich eksperymentach lekkie biblioteki osiągają lepszą wydajność w porównaniu do bardziej rozbudowanych technologii. Wyniki czwartego eksperymentu zostały przedstawione na wykresie 4.



Wykres 4. Średni czas uruchomienia kontenera, zakres prototype

- **Podsumowanie etapu pierwszego**

Porównanie wyników eksperymentów z etapu 1 zostało wykonane poprzez uśrednienie pozycji poszczególnych technologii w rankingach z tego etapu. Przedstawia je tabela 9.

Tabela 9. Ranking technologii na podstawie średniej pozycji w wszystkich eksperymentach z etapu 1

	Pozycja w rankingu
Spring	**1**
Google Guice	3
PicoContainer	5
Weld	1
EasyDI	**6**
Silk DI	4

Takie zestawienie wyników ostatecznie potwierdza zasadność używania mniej rozbudowanych technologii w aspekcie krótkiego czasu startu kontenera. Stosunkowo długie czasy bardziej rozbudowanych technologii mogą być związane z koniecznością wykonywania dodatkowych akcji przed uruchomieniem kontenera.

4.2. Czas wstrzykiwania zależności

Drugi etap testów składał się z pomiaru czasu, jaki potrzebny jest do wstrzyknięcia określonej liczby obiektów w poszczególnych technologiach.

- **Eksperyment 5**

Piąty eksperyment polegał na pomiarze czasu jaki niezbędny jest do wstrzyknięcia określonej liczby obiektów o zakresie Singleton podczas pierwszego uruchomienia programu. Tabela 10 prezentuje otrzymane wyniki.

Tabela 10. Czas pierwszego wstrzyknięcia po zainicjowaniu kontenera, zakres singleton

	Car	Operator (100 obiektów CarShop)	Operator (1000 obiektów CarShop)	Operator (100000 obiektów CarShop)
	Czas wykonania [ms]			
Spring	**0,0006**	**0,0007**	**0,0007**	**0,0007**
Google Guice	0,0688	0,0424	0,3296	2,8463
PicoContainer	0,0595	0,1515	0,5575	3,5203
Weld	0,1280	0,1834	**0,6041**	**3,8999**
EasyDI	**0,2836**	**0,2279**	0,6022	3,7230
Silk DI	0,0064	0,0344	0,3359	3,0075

W etapie drugim wyniki ponownie charakteryzują się dużym zróżnicowaniem. Dodatkowo widać, że ilość wstrzykiwanych obiektów ma niewielki wpływ na rezultaty tego eksperymentu. Ranking uwidaczniający różnice pomiędzy technologiami przedstawia tabela 11.

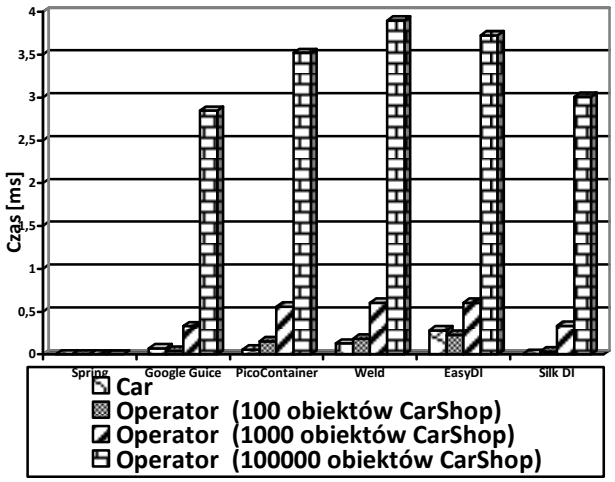
Tabela 11. Ranking technologii na podstawie czasu pierwszego wstrzyknięcia, zakres singleton

	Pozycja w rankingu
Spring	**6**
Google Guice	5
PicoContainer	3
Weld	**1**
EasyDI	2
Silk DI	4

W tym przypadku na najlepszą pozycję wysunęła się najbardziej rozbudowana technologia. Jest to odmienna sytuacja w stosunku do poprzednich eksperymentów, gdzie bezkonkurencyjne były małe biblioteki. Świadczy to o dobrze zaprojektowanym kontenerze, który charakteryzuje się większą wydajnością podczas inicjalizacji obiektów. Na podstawie wyników technologie zostały podzielone na 3 grupy w kolejności od najgorszych do najlepszych:

3) Weld, EasyDI,
4) PicoContainer, Silk DI oraz Google Guice, które są głównie mniej rozbudowanymi technologiami,
5) Spring.

Wyniki piątego eksperymentu zostały przedstawione na wykresie 5.



Wykres 5. Czas pierwszego wstrzyknięcia po zainicjowaniu kontenera, zakres singleton

- **Eksperyment 6**

Kolejny eksperyment polegał na wstrzyknięciu określonej liczby obiektów o zakresie Singleton podczas kolejnych 500 prób uruchomień programu. Jego wyniki zostały zgromadzone w tabeli 12.

Tabela 12. Średni czas wstrzyknięcia jednego obiektu, zakres singleton

	Car	Operator (100 obiektów CarShop)	Operator (1000 obiektów CarShop)	Operator (100000 obiektów CarShop)
	Czas wykonania [ms]			
Spring	0,000052	0,000054	0,000054	0,000071
Google Guice	0,000074	0,000077	0,000069	0,000069
PicoContainer	**0,000021**	**0,000025**	0,000028	**0,000023**
Weld	**0,000746**	**0,000817**	**0,000704**	**0,000851**
EasyDI	0,000023	0,000029	**0,000026**	0,000025
Silk DI	0,000052	0,000054	0,000054	0,000056

Ponownie zauważono znaczne różnice między czasami. Porównując tabele 10 z tabelą 12 można zauważyć, że kolejne wstrzyknięcia są zdecydowanie szybsze. Świadczy to o tym, że podczas pierwszego użycia obiekty muszą być zainicjalizowane. Na podstawie rezultatów z tabeli 12 został stworzony ranking, który przedstawia tabela 13.

Dodatkowo wykonano następujący podział technologii ze względu na otrzymane czasy w kolejności od najgorszych do najlepszych:

1) Weld,

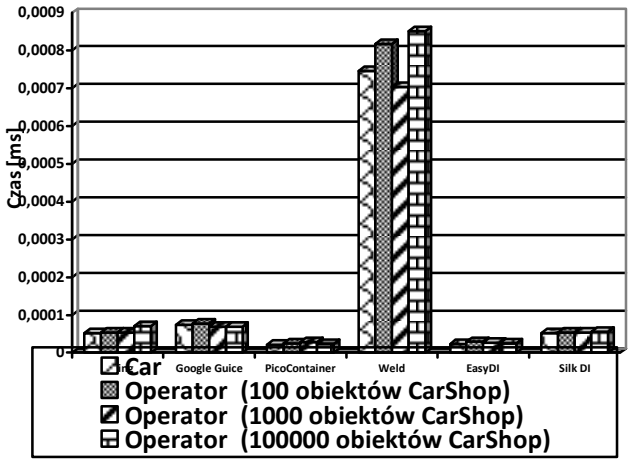
2) Google Guice, Spring oraz Silk DI,
3) EasyDI, PicoContainer.

Tabela 13. Ranking technologii na podstawie średnich czasów wstrzyknięć, zakres singleton

	Pozycja w rankingu
Spring	3
Google Guice	2
PicoContainer	**6**
Weld	**1**
EasyDI	5
Silk DI	4

Tym razem kolejność frameworków ulega zmianie. Przy kolejnych wstrzyknięciach ponownie najlepsze okazują się mniej rozbudowane technologie. Jedyną wyróżniającą się biblioteką jest Weld, który poprzez zastosowanie mechanizmu dynamicznego wstrzykiwania zależności za każdym razem dostarcza nowy obiekt [5]. Ponadto tak jak w przypadku eksperymentu 5, rozmiar obiektów ma niewielki wpływ na osiągnięte czasy.

Wyniki szóstego eksperymentu zostały przedstawione na wykresie 6.



Wykres 6. Średni czas wstrzyknięcia jednego obiektu, zakres singleton

- **Eksperyment 7**

Siódmy eksperyment polegał na pomiarze czasu jaki potrzebny jest do wstrzyknięcia określonej liczby obiektów o zakresie Prototype podczas pierwszego uruchomienia programu. Tabela 14 przedstawia osiągnięte rezultaty.

Z danych otrzymanych z testu w zakresie Prototype widoczne są znaczne różnice pomiędzy czasami, w zależności od ilości wstrzykiwanych obiektów. W tabeli 15 umieszczono ranking technologii na podstawie osiągniętych czasów.

Ponownie zauważono, że najpopularniejsze technologie okazały się najgorsze. Wyróżniono 3 grupy w zależności od technologii w kolejności od najgorszych do najlepszych.

1) Weld,
2) Google Guice, Silk DI oraz PicoContainer,
3) Spring i EasyDI.

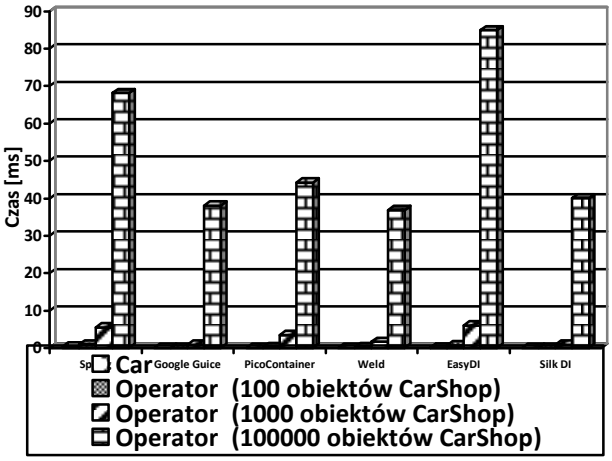
Tabela 14. Czas pierwszego wstrzyknięcia, zakres prototype

	Car	Operator (100 obiektów CarShop)	Operator (1000 obiektów CarShop)	Operator (100000 obiektów CarShop)
	Czas wykonania [ms]			
Spring	**0,3008**	**0,9481**	5,5422	68,1141
Google Guice	0,0541	**0,0094**	0,9062	38,1003
PicoContainer	0,0483	0,2607	3,4635	44,2364
Weld	0,0828	0,1944	1,6446	**36,8724**
EasyDI	0,1173	0,6700	**5,9602**	**84,9424**
Silk DI	**0,0017**	0,0130	**0,8636**	40,0270

Tabela 15. Ranking technologii na podstawie czasu pierwszego wstrzyknięcia, zakres prototype

	Pozycja w rankingu
Spring	2
Google Guice	5
PicoContainer	3
Weld	**6**
EasyDI	**1**
Silk DI	4

Wyniki siódmego eksperymentu zostały przedstawione na wykresie 7.



Wykres 7. Czas pierwszego wstrzyknięcia, zakres prototype

- **Eksperyment 8**

Ostatni eksperyment polegał na wstrzyknięciu określonej liczby obiektów o zakresie Prototype podczas kolejnych 500 prób uruchomień programu. Rezultaty przedstawia tabela 16.

Tabela 16. Średni czas wstrzykiwania jednego obiektu, zakres prototype

	Car	Operator (100 obiektów CarShop)	Operator (1000 obiektów CarShop)	Operator (100000 obiektów CarShop)
	Czas wykonania [ms]			
Spring	**0,001516**	**0,015864**	**0,334080**	**3,898427**
Google Guice	0,000110	0,003328	**0,286363**	**3,073268**
PicoContainer	0,000188	0,003438	0,333198	3,696530
Weld	0,000930	0,004231	0,319661	3,294051
EasyDI	0,000262	**0,002683**	0,313320	3,657439
Silk DI	**0,000049**	0,002879	0,293516	3,106973

Wyniki zgromadzone w tabeli 16 świadczą, że czasy kolejnych wstrzyknięć są zdecydowanie lepsze. Na ich

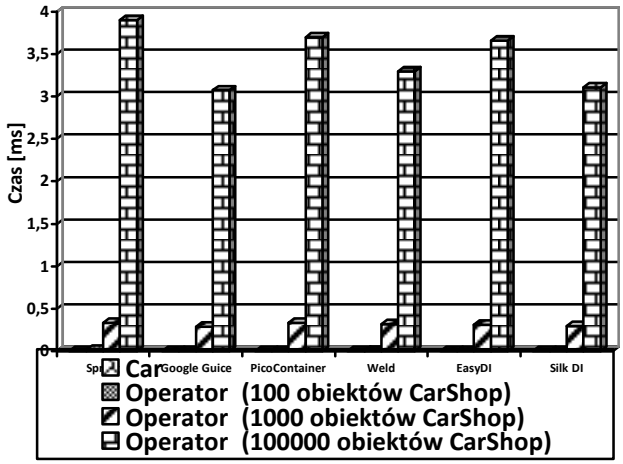
podstawie został stworzony ranking, który przedstawia tabela 17.

Tabela 17. Ranking technologii na podstawie średnich czasów wstrzyknięć, zakres prototype

	Pozycja w rankingu
Spring	1
Google Guice	6
PicoContainer	2
Weld	4
EasyDI	3
Silk DI	5

Grupy podziału względem technologii prezentują się podobnie jak w eksperymencie 7, z wyjątkiem frameworku EasyDI, który podczas pierwszego uruchomienia programu dla dużych obiektów osiągał największe czasy. Jest to spowodowane małą wydajnością tej technologii dla dużych obiektów, co przyczyniło się do jej niskiej pozycji w rankingu.

Wyniki ósmego eksperymentu zostały przedstawione na wykresie 8.



Wykres 8. Średni czas wstrzykiwania jednego obiektu, zakres prototype

- **Podsumowanie etapu drugiego**

Porównanie wyników eksperymentów z etapu 2 zostało wykonane poprzez uśrednienie pozycji poszczególnych technologii w rankingach z tego etapu. Przedstawia je tabela 18.

Tabela 18. Ranking technologii na podstawie średniej pozycji w wszystkich eksperymentach z etapu 2

	Pozycja w rankingu
Spring	1
Google Guice	5
PicoContainer	4
Weld	2
EasyDI	3
Silk DI	5

Również w tym etapie okazało się, że mniej rozbudowane technologie mają przewagę nad bardziej złożonymi narzędziami. Ich relatywnie dobre czasy prawdopodobnie są związane z prostotą mechanizmów użytych w tych frameworkach.

5. Wnioski

Na podstawie przeprowadzonych eksperymentów można zauważyć, że frameworki osiągają różne czasy dla tych samych operacji.

Z badań przeprowadzonych w etapie pierwszym okazało się, że na czas startu kontenera nie ma wpływu zakres widoczności, ani liczba używanych obiektów. Dodatkowo zaobserwowano przewagę lekkich bibliotek, co może być spowodowane bardziej rozbudowanymi mechanizmami w konkurencyjnych bibliotekach oraz koniecznością wykonywania specyficznych akcji przed startem kontenera.

W przypadku etapu drugiego zauważono, że przy wstrzykiwaniu zależności znaczny wpływ mają zakresy widoczności. W zakresie Singleton nie istnieją znaczne różnice w zależności od rozmiaru obiektów. Przeprowadzona analiza wykazuje, że mniejsze biblioteki osiągają lepsze rezultaty od rozbudowanych narzędzi. W każdym przypadku najgorszy okazał się framework Weld. Wyjątek stanowi tutaj eksperyment 1, gdzie najlepszy był Spring. Świadczy to o doskonałym zaprojektowaniu kontenera w tej technologii, co pozwala na szybki dostęp do obiektów o różnych rozmiarach, tuż po jego uruchomieniu. W przypadku zakresu Prototype zaobserwowano wykładniczy przyrost czasu w zależności od rozmiaru wstrzykiwanych obiektów. Jednak w przeciwieństwie do zakresu Singleton wszystkie frameworki działały podobnie (nie było znacznych różnic w czasie).

Ostatecznie dzięki przeprowadzonym eksperymentom widać, że teza stawiana we wstępie okazała się fałszywa. Potwierdziło to zasadność korzystania z małych bibliotek w celu wstrzykiwania zależności.

Literatura

[1] Karami G.,Tian J.: Improving web application reliability and testing using accurate usage models. [W]: Studies in Computational Intelligence, Volume 722, pp. 75-92, 2018.

[2] Bąk T, Sakowicz B.: Development of advanced J2EE solutions based on lightweight containers on the example of "e-department" application. [W]: Proceedings of the International Conference on Mixed Design of Integrated Circuits and Systems, Article number 1706692, pp. 779-782, MIXDES, 2006.

[3] Spring Dependency Injection Styles – Why I love Java based configuration. https://blog.codecentric.de/en/2012/07/spring-dependency-injection-styles-why-i-love-java-based-configuration/ [2017]

[4] Yang H.Y.,Tempero E.: An empirical study into use of dependency injection in Java. [W]: Proceedings of the Australian Software Engineering Conference, ASWEC 4483212, 2008.

[5] Guice documentation. https://github.com/google/guice [15.10.2017]

[6] Spring documentation. https://docs.spring.io/spring/docs/current/spring-framework-reference/html/beans.html [15.10.2017]

[7] PicoContainer documentation. http://picocontainer.com/ [15.10.2017]

[8] Kontekstowe komponenty w J2EE 6 - Wstrzykiwanie zależności. http://michalorman.pl/blog/2009/11/kontekstowe-komponenty-w-j2ee-6-wstrzykiwanie-zaleznosci/ [15.10.2017]

[9] Shi W., Zhang Z.: Design and application of lightweight container. [W]: Jisuanji Gongcheng/Computer Engineering, Volume 32, Issue 20, pp. 65-66, 20 October 2006.

[10] Wstrzykiwanie zależności a testy jednostkowe - złoty środek. https://4programmers.net/Inzynieria_oprogramowania/Wstrzykiwanie_zaleznosci [15.10.2017]

[11] Shrinidhi R., Hudli and Raghu V. Hudli: A Verification Strategy for Dependency Injection. Lecture Notes on Software Engineering, Vol. 1, No. 1, February, 2013.

[12] Weld documentation. https://docs.jboss.org/weld/reference/latest/en-US/html/injection.html [15.10.2017]

[13] EasyDI documentation. https://github.com/lestard/EasyDI [15.10.2017]

[14] Silk DI documentation. http://www.silkdi.com/ [15.10.2017]