JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 26 (2023) 42–47
Received: 20 September 2022
Accepted: 11 October 2022

A comparative analysis of contemporary integrated Java environments

Analiza porównawcza współczesnych zintegrowanych środowisk do pracy w języku Java

Cezary Piotr Kaczorowski

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

Abstract

The purpose of this study was to compare three of the most popular Java development environments: IntelliJ IDEA, Netbeans, and Eclipse, in order to identify the best one for specific user archetypes. Using a program specially designed for this task, the execution time was measured, as well as the CPU and RAM usage during the execution of the program on each environment. The obtained data was collected and processed accordingly. Additionally, the environments were compared in a number of ways, including features that distinguish them from each other, ease learnability for a new user or support from the developers. The study concluded with IntelliJ IDEA being the best environment for all of the considered archetypes.

Keywords: Java environments; IntelliJ IDEA; NetBeans; Eclipse

Streszczenie

Celem pracy było porównanie trzech najpopularniejszych środowisk programistycznych do pracy w języku Java: IntelliJ IDEA, Netbeans i Eclipse, w celu wskazania najlepszego z nich dla konkretnych archetypów użytkowników. Przy pomocy stworzonego specjalnie programu zbadany został czas wykonywania programu, a także obciążenie procesora i pamięci w trakcie wykonywania programu w poszczególnych środowiskach. Dodatkowo środowiska zostały porównane pod wieloma innymi względami, m.in. pod kątem cech odróżniających je od siebie, łatwości opanowywania przez nowego użytkownika, czy wsparcia ze strony twórców oprogramowania. W rezultacie stwierdzono, że IntelliJ IDEA jest najlepszym środowiskiem dla wszystkich rozpatrywanych archetypów użytkowników.

Słowa kluczowe: środowiska Java; IntelliJ IDEA; NetBeans; Eclipse

*Corresponding author

Email address: cezary.kaczorowski@pollub.edu.pl (C. P. Kaczorowski)

1. Wstęp

W obecnych czasach programiści mają do wyboru wiele platform umożliwiających im tworzenie oprogramowania w najróżniejszych językach. Nie inaczej jest z tymi, którzy tworzą aplikację w oparciu o język Java[1, 2]. Dodatkowo częsta aktualizacja platform powoduje duże wahania użyteczności poszczególnych rozwiązań na tle konkurencji.

W związku z mnogością rozwiązań i towarzyszącą im dynamiką zmian, pojawia się potrzeba dostarczenia obiektywnej wiedzy, która pomoże programistom języka Java w wyborze zintegrowanego środowiska programistycznego. Niniejszy artykuł dotyczy porównania środowisk IntelliJ IDEA, NetBeans oraz Eclipse, pod kątem wydajności, funkcjonalności oraz dopasowania do potrzeb typowych archetypów użytkowników. Do porównania wybrano środowiska, które w [3] uznano za najpopularniejsze, a więc warte szczególnej uwagi. Warto wskazać, że nie udało się odnaleźć prac innych autorów, które ten temat analizowałyby w tak szerokim spektrum.

Porównywane środowiska programistyczne zostały ocenione w oparciu o autorskie kryteria uwzględniające archetypy ich potencjalnych użytkowników. Wykonano testy, mające na celu zbadanie czasu wykonywania programu oraz obciążenia komputera generowanego w trakcie wykonywania programu. Dodatkowo przeanalizowano najważniejsze cechy poszczególnych środowisk, ich funkcjonalności oraz cechy odróżniające je od siebie nawzajem.

Postawiono pytanie badawcze: Czy środowisko IntelliJ IDEA jest najlepszym środowiskiem programistycznym do pracy w języku Java? Towarzyszą temu następujące cztery hipotezy:

- H1: Środowisko IntelliJ IDEA jest wydajniejsze niż NetBeans czy Eclipse.
- H2: Środowisko IntelliJ IDEA jest bardziej popularne niż NetBeans czy Eclipse.
- H3: Środowisko IntelliJ IDEA jest najlepiej rozwijanym środowiskiem programistycznym.
- H4: IntelliJ IDEA jest najmniej przystępnym środowiskiem do pracy w języku Java.

2. Metodyka badań

W kolejnych podrozdział przedstawiono aspekty metodyki badań takie, jak: kryteria oceny, archetypy użytkowników, przebieg procedury testowej, sprzęt i oprogramowanie.

2.1. Kryteria oceny

Na potrzeby porównania wybranych środowisk programistycznych języka Java zdefiniowano następujące kryteria oceny:

1. Wydajność – czas w sekundach wykonywania testowego programu w poszczególnych środowiskach.
2. Obciążenie procesora – procentowe obciążenie procesora w trakcie wykonywania testowego programu.
3. Zajętość pamięci – zajętość pamięci RAM w MB w trakcie wykonywania testowego programu.
4. Popularność – zainteresowanie środowiskiem wśród populacji programistów (określone na podstawie zewnętrznych źródeł).
5. Funkcjonalność – mocne i słabe strony wybranych środowisk oraz odróżniające je od siebie możliwości.

2.2. Archetypy użytkowników

Podczas porównania wybranych środowisk programistycznych wspomagano się archetypami użytkowników. Archetypy użytkowników pozwalają zdywersyfikować oczekiwania wobec optymalnego wyboru środowiska, a także ustanowić hierarchię ich oczekiwań. Zdecydowano się zdefiniować następujące archetypy:

1. Zwykły użytkownik – nie ma specjalnych wymagań dotyczących środowiska, wszystkie kryteria stawia na równi.
2. Użytkownik ze słabszym komputerem - potrzebuje środowiska, które będzie najmniej obciążało komputer, mniej uwagi zwraca na szybkość działania i nie przykłada wcale uwagi do popularności czy funkcjonalności środowiska.
3. Doświadczony użytkownik – potrzebuje środowiska, które zaoferuje najwięcej funkcjonalności i nie zwraca uwagi na pozostałe kryteria.

Archetypy w połączeniu z kryteriami oceny umożliwiają wskazanie najlepszego środowiska dla danego użytkownika. Każdy archetyp posiada własną hierarchię potrzeb, w związku z tym najlepsze dla niego środowisko powinno osiągać najlepsze rezultaty w istotnych dla archetypu kryteriach oceny. W przypadku niejednoznacznych wyników dla najistotniejszych kryteriów dla danego archetypu, następne w kolejności będą brane pod uwagę kryteria pokrewne.

2.3. Przebieg procedury testowej

Dla celów przetestowania wybranych środowisk programistycznych został napisany specjalny program. W trakcie jego tworzenia postępowano zgodnie z zasadami pisania dobrego kodu [4, 5]. Zadaniem programu było wygenerowanie odpowiednio dużego obciążenia, aby uzyskać rezultaty mogące podlegać miarodajnej analizie.

Program testowy składał się z trzech testów, które polegały na przeprowadzaniu dużych obliczeń w pętli. Każdy z testów był przeprowadzany przez osobny wątek tak, aby testy mogły wykonywać się jednocześnie. Kod poszczególnych przypadków testowych został przedstawiony na listingach 1-3.

Procedura testowa była wykonywana według następującego schematu:

1. Wstępne uruchomienie programu testowego w aktualnie badanym środowisku – dzięki temu program był identyfikowany przez narzędzie VisualVM.
2. Wybranie pomiaru obciążenia procesora (CPU) i obciążenia pamięci RAM (Memory) w narzędziu VisualVM oraz uruchomienie pomiaru przyciskiem „Start”.
3. Uruchomienie przypadków testowych przedstawionych na listingach 1-3.
4. Odczekanie, aż program testowy się wykona a narzędzie VisualVM zbierze wszystkie potrzebne dane.
5. Zapisanie danych z pomiaru i wyeksportowanie ich do pliku z rozszerzeniem csv.

Listing 1: Kod pierwszego testu

```
public static void firstTest(int index) {
    int n = 10000*index;
    double var1 = 123456789;
    double var2 = 987654321;
    for(int j = 0; j < n; j++) {
        for(int i = 0; i < n; i++) {
            var1 = var1 * var2;
        }
    }
    System.out.println("\n---First test progress: "
            + index +  "0%");
}
```

Listing 2: Kod drugiego testu

```
public static void secondTest(int index) {
    int n = 10000*index, var = 0;
    String chain = "";
    List<String> list = new ArrayList<>();
    for(int i = 0; i < n; i++) {
        var = (int) pow(i,i);
        list.add(Integer.toString(var));
        chain += list.get(i);
    }
    chain = Double.toString(sqrt(
            Double.parseDouble(chain)));
    if(Double.parseDouble(chain)>var) {
        chain = Double.toString(pow(var,var));
    }
    System.out.println("\n---Second test progress: "
            + index +  "0%");
}
```

Listing 3: Kod trzeciego testu

```
public static void thirdTest(int index) {
    int n = 10000*(index-5);
    for(int j = n; j > 0; j-=2) {
        for(int i = 0; i < n; i+=2) {
            Double var = pow(i,j);
        }
    }
    System.out.println("\n---Third test progress: "
            + index +  "0%");
}
```

2.4. Sprzęt i oprogramowanie

Parametry komputera, na którym przeprowadzono testy były następujące:

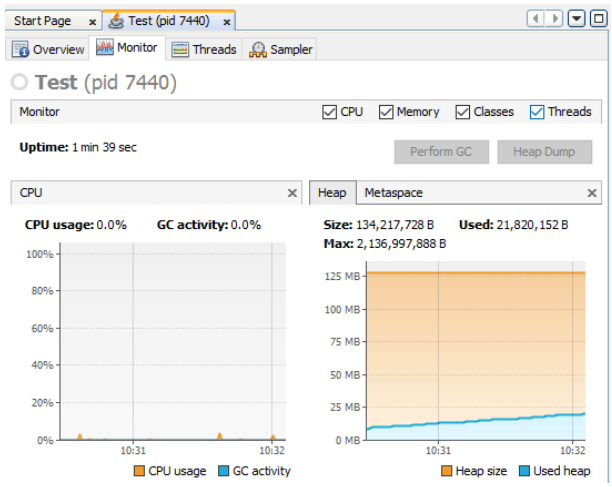
- Procesor: Intel Core i7-9700 3.00GHz.

- Karta graficzna: NVIDIA GeForce RTX 2060.
- Pamięć RAM: 32GB DDR4.
- System operacyjny: Windows 10 wersja 64 bitowa.
- Wersja środowiska Java: JDK 11[6].

Porównywane środowiska programistyczne języka Java były zainstalowane w następujących wersjach:

- Wersja IntelliJ IDEA: 2022.1.1.
- Wersja NetBeans: IDE 13.
- Wersja Eclipse: 2022-03(4.23.0).

W celu zebrania danych odnośnie czasu wykonywania programu testowego, obciążenia procesora i zajętości pamięci RAM, wykorzystano narzędzie VisualVM [7], które współpracuje z każdym z porównywanych środowisk programistycznych. Rysunek 1 przedstawia przykładowy pomiar przy użyciu tego programu.



Rysunek 1: Przykładowy zrzut ekranu, prezentujący pomiary w narzędziu VisualVM dla środowiska Eclipse.

3. Analiza wyników testów

Dla każdego pomiaru zebrano następujące dane dla każdego z porównywanych środowisk:

- czas wykonania programu [s],
- minimalne obciążenie procesora [%],
- maksymalne obciążenie procesora [%],
- średnie obciążenie procesora [%],
- mediana obciążenia procesora [%],
- minimalna zajętość pamięci [MB],
- maksymalna zajętość pamięci [MB],
- średnia zajętość pamięci [MB],
- mediana zajętości pamięci [MB].

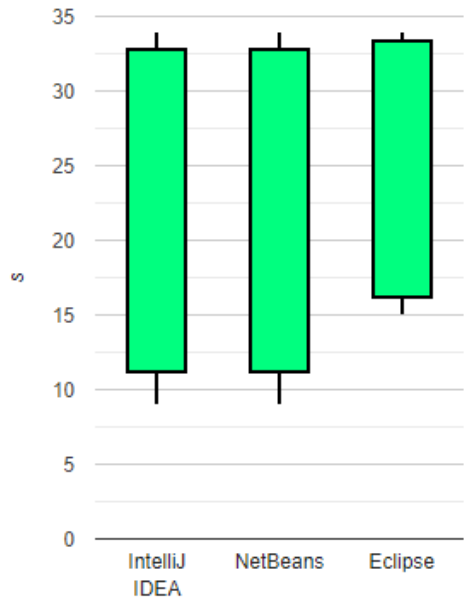
3.1. Czas wykonywania programu

Zestawienie czasów wykonywania programu testowego przedstawiono w Tabeli 1. Środowisko Eclipse okazało się najwolniejszym z porównywanych środowisk. Środowiska Netbeans i IntelliJ IDEA uzyskały bardzo zbliżone wyniki, z nieznaczną przewagą środowiska Netbeans.

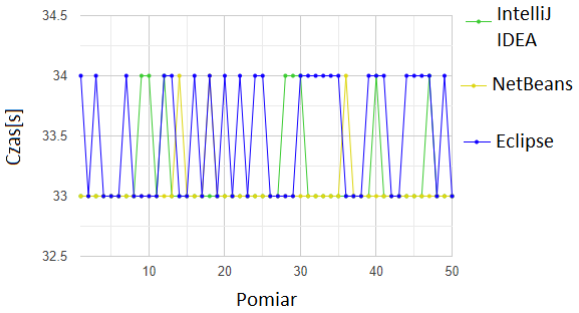
Tabela 1: Czas wykonywania programu testowego [s]

	IntelliJ IDEA	NetBeans	Eclipse
Minimum	9,00	9,00	15,00
Maksimum	34,00	34,00	34,00
Mediana	23,50	23,00	26,00
Średnia	33,27	33,19	33,66

Na Rysunku 2 przedstawiono wykres pudełkowy dla zebranych czasów wykonywania programu testowego. Z kolei Rysunek 3 przedstawia czasy wykonania poszczególnych pomiarów.



Rysunek 2: Wykres pudełkowy dla czasu wykonywania programu testowego [s].



Rysunek 3: Wykres liniowy dla czasu wykonywania programu testowego [s].

Wszystkie czasy różnią się od siebie o bardzo małe wartości, które mogą być bardziej widoczne przy bardziej złożonych i skomplikowanych projektach. Jednakże z zebranych danych wynika, że różnica pomiędzy dwoma wiodącymi w tej kategorii środowiskami jest marginalna, w związku z czym zarówno NetBeans, jak i IntelliJ IDEA mogą być uznane za równie szybkie.

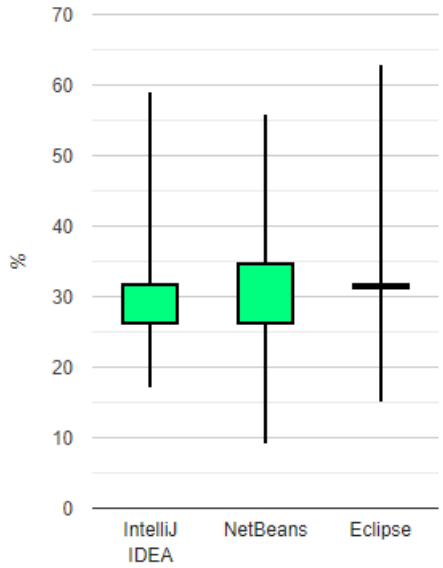
3.2. Obciążenie procesora

Zestawienie obciążenia procesora podczas wykonywania programu testowego przedstawiono w Tabeli 2. Środowisko Eclipse uzyskało zdecydowanie najgorszy wynik. Środowiska Netbeans i IntelliJ IDEA ponownie otrzymały zbliżone wyniki, z przewagą na rzecz tego pierwszego.

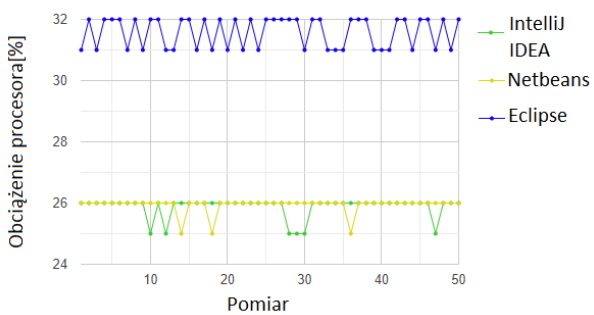
Na Rysunku 4 przedstawiono wykres pudełkowy dla zebranych obciążeń procesora podczas wykonywania programu testowego. Z kolei Rysunek 5 przedstawia obciążenia procesora podczas poszczególnych pomiarów.

Tabela 2: Obciążenie procesora podczas wykonywania programu testowego [%]

	IntelliJ IDEA	NetBeans	Eclipse
Minimum	17,00	9,00	15,00
Maksimum	59,00	56,00	63,00
Mediana	26,00	26,00	31,00
Średnia	26,32	26,32	32,07



Rysunek 4: Wykres pudełkowy dla obciążenie procesora podczas wykonywania programu testowego [%].



Rysunek 5: Wykres liniowy dla obciążenia procesora podczas wykonywania programu testowego [%].

Okazało się, że środowisko Eclipse najbardziej obciążało procesor komputera. Ponownie, jak w przypadku czasu wykonania, różnica pomiędzy IntelliJ IDEA i NetBeans jest na tyle niewielka, że oba środowiska można uznać za równie wydajne w kwestii obciążenia procesora.

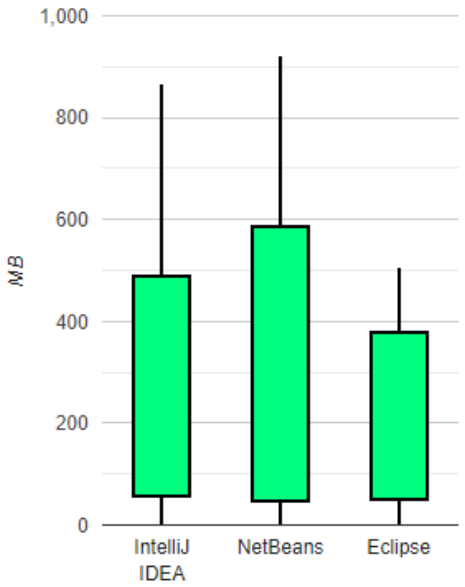
3.3. Zajętość pamięci

Zestawienie zajętości pamięci podczas wykonywania programu testowego przedstawiono w Tabeli 3. Tym razem środowisko Eclipse uzyskało zdecydowanie najlepszy wynik. Środowiska Netbeans i IntelliJ IDEA ponownie otrzymały zbliżone wyniki, z przewagą na rzecz tego drugiego.

Tabela 3: Tabela zawierająca najważniejsze wartości z wykresu pudełkowego średniego zużycia pamięci [MB]

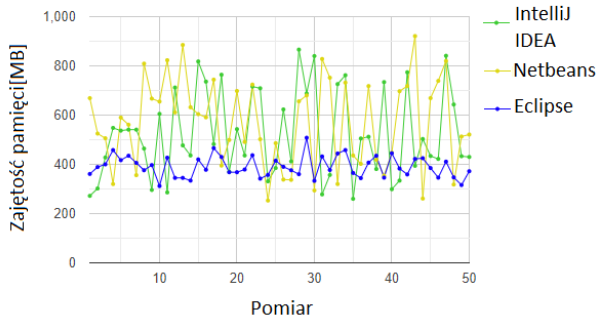
	IntelliJ IDEA	NetBeans	Eclipse
Minimum	13,21	7,54	14,13
Maksimum	866,00	921,00	508,00
Mediana	178,50	174,00	205,00
Średnia	524,19	570,15	391,43

Na Rysunku 6 przedstawiono wykres pudełkowy dla zebranych zajętości pamięci podczas wykonywania programu testowego. Z kolei Rysunek 7 przedstawia zajętość pamięci podczas poszczególnych pomiarów.



Rysunek 6: Wykres pudełkowy dla zajętości pamięci podczas wykonywania programu testowego [MB].

Zwycięzcą w kategorii zajętości pamięci okazało się środowisko Eclipse. Na kolejnym miejscu uplasowało się środowisko IntelliJ IDEA, a następnie NetBeans. Tym razem różnice okazały się znaczące.



Rysunek 7: Wykres liniowy dla zajętości pamięci podczas wykonywania programu testowego [MB].

4. Porównanie funkcjonalności środowisk programistycznych

W kolejnych podrozdziałach przedstawiono analizę funkcjonalności porównywanych środowisk programistycznych.

4.1. Eclipse

Eclipse [8] jest darmowym środowiskiem programistycznym, a więc jest ogólnodostępny dla wszystkich użytkowników. Posiada bogatą dokumentację, która ułatwia nowym użytkownikom nabycie wiedzy, jak korzystać z tego środowiska. Eclipse dostarcza wsparcie dla platformy GitHub, jednakże nie jest ono zbyt intuicyjne w obsłudze. Jest również dostępna opcja integracji projektu z narzędziami Docker czy Kubernetes. Eclipse posiada bogatą bibliotekę wtyczek (ang. plug-in) wspierających i ułatwiających pracę programisty. Zgodnie z danymi z Eclipse Marketplace jest ich ponad 1500. Co jednak odróżnia Eclipse od pozostałych środowisk, to wsparcie dla tworzenia własnych języków w oparciu o te już istniejące - opisano to chociażby w artykułach [9] i [10]. Choć jest to zdecydowanie ogromna zaleta tego środowiska, to jednak opcja tworzenia własnego języka nie jest do końca przydatna dla programisty, który zamierza pracować z językiem Java. Eclipse jest aktualizowany co 3 miesiące.

4.2. NetBeans

Środowisko Netbeans [11] tak, jak i Eclipse, jest w pełni darmowym zintegrowanym środowiskiem programistycznym. Jest ono dosyć proste, a więc względnie łatwe w nauce, jednak dla bardziej doświadczonych użytkowników nie oferuje zbyt wiele, zwłaszcza na tle pozostałych dwóch środowisk. Netbeans nie oferuje funkcjonalności, która by go szczególnie wyróżniała na tle konkurencji w postaci Eclipse czy IntelliJ IDEA. Środowisko posiada mało wtyczek (około 60) i nie oferują one niczego, czego nie oferowałyby pozostałe środowiska. Netbeans posiada wbudowany profiler oparty o VisualVM służący do analizy aplikacji, jednakże nie jest zbyt rozbudowany. Pozostałe środowiska oferują, choćby dzięki dodatkowym wtyczkom, znacznie więcej opcji. Jest dostępne wsparcie dla Git i konteneryzacji. Netbeans jest aktualizowany co około 3-4 miesiące

4.3. IntelliJ IDEA

Środowisko IntelliJ IDEA [12] jest dostępne w dwóch różnych wersjach: darmowej tzw. Community Edition i płatnej tzw. Ultimate Edition. Choć mogłoby się wydawać, że największe zalety tego środowiska będą niedostępne dla większości użytkowników poprzez zablokowanie wielu funkcjonalności w darmowej wersji, to sytuacja ma się inaczej. Znaczna większość funkcji oferowanych przez to środowisko jest dostępna w wersji Community [13]. IntelliJ IDEA posiada około 6035 wtyczek, z czego 5555 jest dostępne w darmowej wersji środowiska. Wyszukiwanie wtyczek jest bardzo intuicyjne, ze względu na wyszukiwarkę, która bardzo dobrze radzi sobie ze słowami kluczowymi. Takie rozwiązanie pozwala łatwo zaspokoić potrzeby projektów, ale może być przytłaczające dla początkującego użytkownika. IntelliJ IDEA, podobnie jak Eclipse, pozwala na integrację z platformą GitHub, ale zawiera ona o wiele więcej opcji i czytelniejszy interfejs. Wśród wielu użytecznych funkcjonalności można wyróżnić m.in.: inteligentne uzupełnianie kodu, generowanie diagramów na podstawie kodu, wyszukiwanie duplikatów kodu, czy integrację kodu w innym języku w projekcie. IntelliJ IDEA posiada wsparcie dla narzędzi Docker i Kubernetes, jednak ta druga jest dostępna jedynie w wersji Ultimate. Inną funkcjonalnością dostępną jedynie w płatnej wersji środowiska są „profiling tools" służące do analizy kodu. Największą wadą tego środowiska, oprócz zablokowania części funkcjonalności w bezpłatnej wersji, jest niezbyt dobra dokumentacja, która sprawia, że nauczenie się korzystania z tego środowiska nie jest zbyt łatwe. IntelliJ IDEA jest aktualizowany co miesiąc.

5. Podsumowanie

Przeprowadzone badania dostarczyły danych pozwalających na wyłonienie najlepszych środowisk w poszczególnych kryteriach:

1. Wydajność: Najlepszymi środowiskami okazały się zarówno NetBeans, jak i IntelliJ IDEA.
2. Obciążenie procesora: Najlepszymi środowiskami okazały się zarówno NetBeans, jak i IntelliJ IDEA.
3. Zajętość pamięci: Najlepszym środowiskiem okazał się bezsprzecznie Eclipse.
4. Popularność: Zgodnie z zewnętrznymi badaniami [3] Eclipse jest obecnie najpopularniejszym środowiskiem do pracy w języku Java.
5. Funkcjonalność: Najlepszym środowiskiem okazało się IntelliJ IDEA, które nawet w darmowej wersji oferuje o wiele więcej funkcjonalności wspomagających pracę programisty niż pozostałe środowiska, a w wersji płatnej jest bezkonkurencyjnie najlepszym środowiskiem pod względem oferowanych unikalnych funkcjonalności.

Po nałożeniu rezultatów w powyższych pięciu kryteriach na archetypy użytkowników, można wydać następujące rekomendacje:

- Zwykły użytkownik: Z racji tego, iż ten użytkownik nie ma żadnej hierarchii wśród kryteriów, najodpowiedniejszym dla takiego użytkownika środowi-

skiem wydaje się to, które okazało się najlepsze w największej liczbie kryteriów. Najlepszym więc wyborem dla zwykłego użytkownika jest IntelliJ IDEA, które jest najlepsze pod względem funkcjonalności oraz ex aequo najlepsze pod względem wydajności i obciążenia procesora.

- Użytkownik ze słabszym komputerem: Zgodnie z preferencjami tego użytkownika najlepszym wyborem będzie środowisko, które generuje najmniejsze obciążenie procesora oraz zajętość pamięci. Jednakże w obu tych kryteriach najlepsze okazują się różne środowiska. W takim wypadku trzeba rozstrzygnąć to za pomocą pokrewnego kryterium pomocniczego. Po wzięciu kryterium wydajności okazuje się, że najlepszym wyborem dla użytkownika ze słabszym komputerem jest IntelliJ IDEA bądź NetBeans.
- Doświadczony użytkownik: W tym przypadku nie ma problemu z wyłonieniem najlepszego środowiska. Jest nim bezsprzecznie IntelliJ IDEA, który okazał się bezsprzecznie najlepszy w tym jednym, ale znaczącym dla podanego archetypu kryterium.

Po przeprowadzeniu analiz i porównań można odpowiedzieć twierdząco na pytanie badawcze, że środowisko IntelliJ IDEA jest najlepszym środowiskiem programistycznym do pracy w języku Java. Można stwierdzić, że IntelliJ IDEA jest wydajniejsze niż pozostałe porównywane środowiska, jest najlepiej rozwijanym środowiskiem, jest najmniej przystępnym środowiskiem, nie jest najbardziej popularnym środowiskiem. Pomimo pewnych negatywnych aspektów, IntelliJ IDEA okazało się preferowanym środowiskiem dla każdego z trzech archetypów użytkowników.

Literatura

[1] B. Eckel, Thinking in Java, Wydanie IV Helion, 2006.

[2] C. S. Horstmann, Java. Podstawy, Wydanie X Helion, 2010.

[3] Ranking najlepszych środowisk IDE Java i kompilatorów języka Java, https://myservername.com/top-10-best-java-ides-online-java-compilers, [22.05.2022].

[4] N. Wirth, Algorytmy + struktury danych = programy, Wydawnictwa Naukowo-Techniczne, 2002.

[5] R. C. Martin, Czysty kod. Podręcznik dobrego programisty, Helion, 2014.

[6] Dokumentacja języka Java w wersji JDK 11: https://docs.oracle.com/en/java/javase/11/docs/api/index.html, [22.05.2022].

[7] Strona główna programu VisualVM: https://visualvm.github.io, [22.05.2022].

[8] Strona Eclipse zawierająca opis środowiska dla deweloperów i najważniejszych funkcjonalności, https://scand.com/company/blog/eclipse-ide-for-java-developers/, [22.05.2022].

[9] A. Margheri, M. Masi, R. Pugliese, F. Tiezzi, A rigorous framework for specification, analysis and enforcement of access control policies, IEEE Transactions on Software Engineering, 45(2017), 2-33.

[10] L. Lopes, F. Martins, A safe-by-design programming language for wireless sensor networks, Journal of Systems Architecture, 63, (2016), 16-32.

[11] Strona NetBeans zawierająca opis środowiska dla deweloperów i najważniejszych funkcjonalności, https://netbeans.apache.org/download/nb120/, [22.05.2022].

[12] Strona IntelliJ IDEA zawierająca opis środowiska dla deweloperów i najważniejszych funkcjonalności, https://www.jetbrains.com/idea/features/, [22.05.2022].

[13] Strona IntelliJ IDEA zawierająca porównanie wersji płatnej (Utimate) z wersją darmową (Community), https://www.jetbrains.com/products/compare/?product=idea&product=idea-ce, [22.05.2022].