ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2019, № 6 (84)

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ ТА МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ

УДК 004.43

І. М. СТОРЧАК[1*], О. П. ІВАНОВ[2*]

[1*]Каф. «Комп'ютерні інформаційні технології», Дніпровський національний університет залізничного транспорту імені академіка В. Лазаряна, вул. Лазаряна, 2, Дніпро, Україна, 49010, тел. +38 (056) 373 15 35,
ел. пошта storchakigor1@gmail.com, ORCID 0000-0002-8434-9765
[2*]Каф. «Комп'ютерні інформаційні технології», Дніпровський національний університет залізничного транспорту імені академіка В. Лазаряна, вул. Лазаряна, 2, Дніпро, Україна, 49010, тел. +38 (056) 373 15 35, ел. пошта iva.dnp@gmail.com, ORCID 0000-0003-1259-6377

АНАЛІЗ МЕХАНІЗМІВ ТА ЕФЕКТИВНОСТІ СПЕЦІАЛІЗОВАНИХ МОВ ФУНКЦІОНАЛЬНОГО ПРОГРАМУВАННЯ

Мета. Автори ставлять за мету встановити відмінності функціональних мов програмування, з`ясувати можливості найбільш популярних мов шляхом їх порівняння та аналізу. Для виявлення основних можливостей мов потрібно розглянути їх структури даних, а також сфери застосування. Виконати аналіз та порівняння прикладів із різних сфер використання мов за метриками складності текстів програм. **Методика**. Відібрано п'ять найпопулярніших спеціалізованих функціональних мов програмування: Erlang, Lisp, F#, Scala та Haskel. Для отримання інформації про можливості кожної мови вивчено їх структури даних, а також сфери застосування, проведено огляд офіційної документації. Експериментальну базу дослідження сформовано з текстів сучасних програмних систем, отриманих із відкритого джерела та підібраних за схожими напрямами застосування й однаковим обсягом тексту. Порівняльний аналіз прикладів програм виконано за метриками Холстеда, які розраховують за допомогою спеціально розробленого програмного забезпечення. Аналіз отриманих оцінок якості виконано графічним способом. **Результати.** Розроблено програмне забезпечення, яке дозволяє отримати метрики Холстеда для вхідних текстів програм на таких мовах функціонального програмування, як Erlang, Lisp, F# та Scala. Складність синтаксису мови програмування Haskel не дозволила використати метрики для оцінки тексту, тому було проведено тільки огляд можливостей за документацією. За допомогою порівняльного аналізу показано відмінність мов та окреслено сфери їх використання. Виконано порівняння прикладів різного об'єму з таких сфер використання, як задачі системного програмування, робота з графікою, математичні розрахунки, системи штучного інтелекту, веб-програмування тощо. **Наукова новизна.** Автори вперше провели порівняльний аналіз спеціалізованих мов за допомогою метрик складності текстів, який дозволив встановити, що мова Lisp має найменший словник і довжину коду, текст на Scala має найбільш структурований вигляд, а F# та Erlang відзначаються зайвою багатослівністю. **Практична значимість.** Отримані висновки та виміри допоможуть під час вибору найбільш ефективної мови функціонального програмування для вирішення конкретних завдань з урахуванням відмінностей у сферах застосування. Розроблене програмне забезпечення дозволяє виконувати виміри для різних текстів програм під час розробки та супроводу складних програмних систем.

*Ключові слова***:** функціональне програмування; метрики Холстеда; можливості мов; порівняння мов; спеціалізовані функціональні мови; Erlang; Haskel; Lisp; F#; Scala

Вступ

Мови програмування класифікують за стилями, які утворюють парадигмами програмування. Загалом поширені такі парадигми, як процедурна, об'єктно-орієнтована та функціональна [14]. Процедурні та об'єктно-орієнтовані парадигми мутують або змінюють дані під час виконання програми. Об'єктно-орієнтована парадигма також заснована на

doi: https://doi.org/10.15802/stp2019/195581

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2019, № 6 (84)

представленні програми у вигляді сукупності об'єктів, кожен із яких є екземпляром певного класу, а класи утворюють ієрархію спадкування [10]. Це дозволяє проектувати і створювати досить складні системи.

З іншого боку, у чистому функціональному стилі дані не існують самі по собі або незалежно [7, 12]. Композиція функціональних викликів із набором аргументів генерує кінцевий результат. У цілому, функціональне програмування розглядає функції й дані як незмінні об'єкти. Функції приймають дані й повертають їх перетвореними або якийсь інший тип даних. У функціональному програмуванні функція ніколи не змінює вхідні дані або стан програми, функції повинні перетворювати дані [11, 13]. Такі властивості дозволяють складати надійні програми, з меншим об'ємом тексту. Структура функціональних програм дозволяє простіше виконувати їх у паралельному та розподіленому режимі.

Мета

Таким чином, виникає проблема аналізу механізмів функціональних мов та порівняння їх можливостей і ефективності в тій чи іншій сферах застосування. Автори ставлять за мету проаналізувати тексти програм на обраних функціональних мовах, виконати їх порівняння за метриками Холстеда. Джерелом для прикладів програм обрано систему збереження відкритого програмного забезпечення GitHub [8].

Методика

Для порівняльного аналізу було обрано п'ять найпопулярніших мов функціонального програмування, для них розглянуто структуру та виконано розрахунок метрик:

– Haskel – стандартизована функціональна мова програмування загального призначення. Є однією з найбільш популярних мов програмування з підтримкою відкладених обчислень [1, 13];

– Erlang – функціональна мова програмування з сильною динамічною типізацією, призначена для створення розподілених обчислювальних систем [6];

– Lisp – сімейство мов програмування, програми й дані в яких представлено системами лінійних списків символів. Для порівняння використано діалект Common Lisp [3];

– F# – це мультипарадигмальна мова програмування з сімейства мов NET Framework, що підтримує функціональне програмування разом з імперативним (процедурним) та об'єктно-орієнтованим програмуванням [5];

– Scala – мультипарадигмальна мова програмування, спроектована короткою й безпечною до типів, для простого і швидкого створення компонентного програмного забезпечення, що поєднує можливості функціонального й об'єктно-орієнтованого програмування [2].

Розглянемо порівняння функціональних можливостей представлених мов (табл. 1).

Таблиця 1

Порівняння функціональних можливостей мов

Table 1

Comparison of language functionality

Функціональна можливість	Мова				
	Haskell	Erlang	Lisp	F#	Scala
Декларації чистоти функцій	+	–	–	–	–
Функції першого класу	+	+	+	+	+
Анонімні функції	+	+	+	+	+
Лексичні замикання	+	+	+	+	+
Часткове застосування	+	–	–	+	+
Каринг	+	–	–	+	+

doi: https://doi.org/10.15802/stp2019/195581

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2019, № 6 (84)

Типи, структури даних і функціональні можливості мов:

1. Декларації чистоти функції. У мовах програмування чиста функція – це функція, яка:

– є детермінованою – має можливість повернення різних значень незважаючи на те, що їй передаються на вхід однакові значення вхідних аргументів;

– не володіє побічними ефектами – наділена можливістю в процесі виконання своїх обчислень до модифікації значення глобальних змінних, реагування на виняткові ситуації та виклик їх обробників.

2. Функції першого класу – це об'єкти першого класу, тобто елементи, які можуть бути передані як параметр або повернуті з функції.

3. Анонімні функції – особливий вид функцій, які оголошують у місці використання. Вони не отримують унікальний ідентифікатор для доступу до них, тобто не можуть бути викликані за посиланням або на ім'я.

4. Лексичні замикання – це можливість використовувати локальну або лямбда-функцію за межами функції-контейнера з автоматичним збереженням контексту (локальних змінних) останньої.

5. Часткове застосування – можливість у ряді мов програмування зафіксувати частину аргументів багатомісної функції і створити іншу функцію з меншою кількістю аргументів.

6. Каринг – перетворення функції від багатьох аргументів на набір функцій, кожна з яких є функцією від одного аргумента.

Із табл. 1 видно, що декларації чистоти функцій підтримує тільки Haskell. Функції першого класу, анонімні функції та лексичні замикання підтримують усі представлені мови, у той час як часткове застосування й каринг не підтримують тільки Erlang та Lisp.

Розглянемо типи і структури даних які підтримують мови (табл. 2).

Таблиця 2

Типи і структури даних функціональних мов

Table 2

Types and structures of functional languages data

Типи і структури даних	Мова				
	Haskell	Erlang	Lisp	F#	Scala
Кортежі	+	+	+	+	+
Алгебраїчні типи даних	+	–	–	+	+
Багатовимірні масиви	+	–	+	+	+/–
Динамічні масиви	–	–	+	+/–	+
Цикл foreach	+	+/–	+	+	+
Спискові включення	+	+	+	+	+
Цілі числа довільної довжини	+	+	+	+	+
Цілі числа з контролем границь	–	–	+	–	–

Знак «+/–» означає, що підтримка здійснюється за допомогою додавання сторонніх бібліотек, модулів або інших механізмів.

1. Кортежі передбачають можливість повернути з функції / методу кортеж (tuple) – неіменований типу даних, що містить кілька безіменних полів довільного типу.

2. Алгебраїчні типи даних – в інформатиці це найбільш загальний складовий тип, який представляє собою тип-суму з типів-творів.

3. Багатовимірні масиви – так звані «масиви з масивів», тобто масив, який містить у собі ще один або кілька масивів.

4. Динамічні масиви характеризуються тим, що масив здатний змінювати свій розмір під час виконання програми.

doi: https://doi.org/10.15802/stp2019/195581

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2019, № 6 (84)

5. Цикл foreach – спеціальний цикл для зручного перебору всіх елементів колекції (списку, масиву, словника та ін.).

6. Спискові включення (абстракція списків, або спискові включення) у синтаксисі деяких мов програмування – це спосіб компактного описання операцій обробки списків.

7. Цілі числа довільної довжини – підтримка цілих чисел необмеженої розрядності, тобто можливість записати як завгодно велике ціле число за допомогою літерала.

8. Цілі числа з контролем границь – можливість визначити тип, значеннями якого можуть бути цілі числа тільки певного інтервалу, наприклад, –10–20, при цьому привласнення змінній значення, що виходить за зазначені рамки, має викликати помилку.

Виходячи з табл. 2, кортежі, цикл foreach, спискові включення й цілі числа довільної довжини підтримують усі представлені мови. Erlang і Lisp не підтримують алгебраїчні типи даних, тому що в динамічних мовах цей механізм не має сенсу.

Багатовимірні масиви не підтримує тільки Erlang. Динамічні масиви не підтримують Haskell та Erlang. Цілі числа з контролем границь підтримує тільки Lisp, що трохи виділяє цю мову серед інших.

Розглянемо, для яких завдань застосовують вибрані мови (табл. 3).

Таблиця 3

Класи завдань, які вирішують функціональні мови

Table 3

Classes of tasks that functional languages solve

Класи завдань	Мова				
	Haskell	Erlang	Lisp	F#	Scala
Системне програмування	+	–	+	–	–
Робота з графікою	+	+/–	+/–	+/–	–/+
Веб-програмування	+	+/–	–/+	+	+
Математичні розрахунки	+	+/–	+/–	++	+
Завдання штучного інтелекту	+/–	++	++	+/–	+/–
Багатопотокове програмування	+	++	+	+	+
Робота з текстом	+	–	+	+	+
Робота з базами даних	+	+	+	+	+
Розробка ігор	+	+	+	–	+/–
Мови й компілятори	+	–	–	+	–
Кросплатформеність	–/+	–	+/–	–/+	+/–

Знак «+/–» означає, що мова підтримує цей напрям завдань, але практично не застосовується в ньому через складність реалізації або непопулярність. Знак «–/+» означає, що мова не підтримує обраний напрям завдань, але за допомогою сторонніх бібліотек і фреймворку, це стає можливим. Знак «++» означає, що мова є найбільш ефективною в цій категорії.

doi: https://doi.org/10.15802/stp2019/195581

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2019, № 6 (84)

Знаки «+/–» і «–/+» указують на те, що використання вибраної мови не рекомендовано в цьому напрямі завдань, тому варто вибрати більш відповідну мову.

З усіх мов тільки Haskel та Erlang є строго функціональними мовами, решта ж мов є мультипарадигмальними, тобто вони підтримують функціональний, об'єктно-орієнтований та інші підходи програмування, що робить їх більш універсальними.

Під час аналізу виявлено, що загалом усі мови підтримують описані в табл. 2 структури даних, у деяких випадках за допомогою сторонніх бібліотек або механізмів.

Хоча мова Haskelli є строго функціональною, її використовують у досить багатьох областях, від роботи з базами даних і графічними інтерфейсами до ігор, інтернет–додатків і системного програмування. Вона також досить популярна в області фінансового програмування, аналізу ризиків, а також у системах підтримки прийняття рішень.

Мову Erlang, хоча й розроблену в основному для застосування в розподілених, стійких до відмов і паралельних системах реального часу, активно використовують в інтернет технологіях, зокрема під час створення серверів, також вона має популярність під час розробки хмарних систем.

Для порівняння прикладів програм використано прості й інформативні метрики Холстеда[9], які належать до кількісних метрик і метрик складності. Обчислюють їх на підставі аналізу числа рядків і синтаксичних елементів вихідного тексту програми.

Основу метрик Холстеда складають шість вимірюваних характеристик програми:

– NUOprtr (Number of Unique Operators) – число унікальних операторів програми, що включає символи-роздільники, імена процедур і знаки операцій (словник операторів);

– NUOprnd (Number of Unique Operands) – число унікальних операндів програми (словник операндів);

– Noprtr (Number of Operators) – загальне число операторів у програмі;

– Noprnd (Number of Operands) – загальне число операндів у програмі;

– TNUOprtr (Theoretical Number of Unique Operators) – теоретичне число унікальних операторів програми;

– TNUOprnd (Theoretical Number of Unique Operands) – теоретичне число унікальних операндів програми.

З огляду на введені позначення можна визначити такі формули:

– словник програми:

$$PD = NUOprtr + NUOprnd;$$

– теоретичний словник програми:

$$TPD = TNUOprtr + TNUOprnd;$$

– довжина програми:

$$PL = Noprtr + Noprnd;$$

– теоретична довжина програми:

$$TPL = NUOprtr \cdot \log_2(NUOprtr) + NUOprnd \cdot \log_2(NUOprnd);$$

– об'єм програми:

$$PS = PL \ \log_2 PD;$$

– теоретичний об'єм програми:

$$TPS = TPL \ \log_2 TPD \ ;$$

– рівень якості програми:

$$Pml = \frac{(2NUOprnd)}{NUOprtr \cdot Noprnd};$$

– рівень якості програмування:

$$PnQL = \frac{TPS}{PS};$$

– складність розуміння програми:

$$DUP = \frac{PS}{PnQL};$$

– трудомісткість кодування програми:

$$CCP = \frac{1}{PmQL};$$

– оцінка необхідних інтелектуальних зусиль:

doi: https://doi.org/10.15802/stp2019/195581

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2019, № 6 (84)

$$ANI = TPL \log_2\left(\frac{PD}{PnQL}\right).$$

Результати

Для обчислення метрик Холстеда розроблено спеціальне програмне забезпечення. Приклади програм узяті з інтернет-ресурсу GitHub [8] – це найбільший веб-сервіс для хостингу IT-проектів, який налічує тисячі проектів із відкритим вихідним кодом.

Аналіз метрик проведено тільки для чотирьох із вищеописаних мов, мова Haskell має дуже специфічний синтаксис, який важко піддається обробці. Отримані результати представлено за допомогою діаграм, які розбиті на два блоки. Приклади коду не наведено, оскільки вони мають великий об’єм.

Результати проведеного порівняння коду програм для обчислення числа Фібоначчі наведено на рис. 1 і 2.

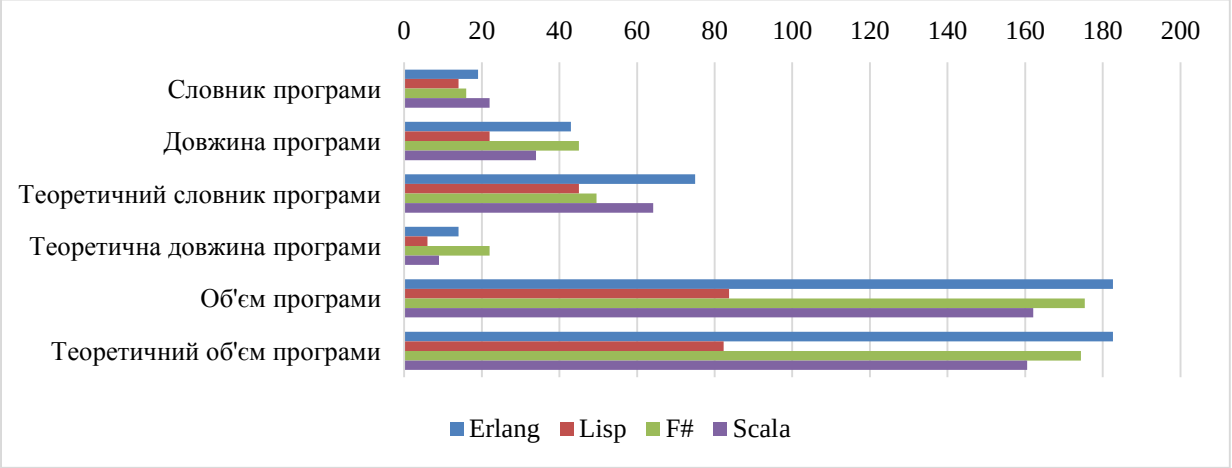

Рис. 1. Характеристики програм об’ємом менше 30 рядків коду

Fig. 1. Program features with less than 30 lines of code

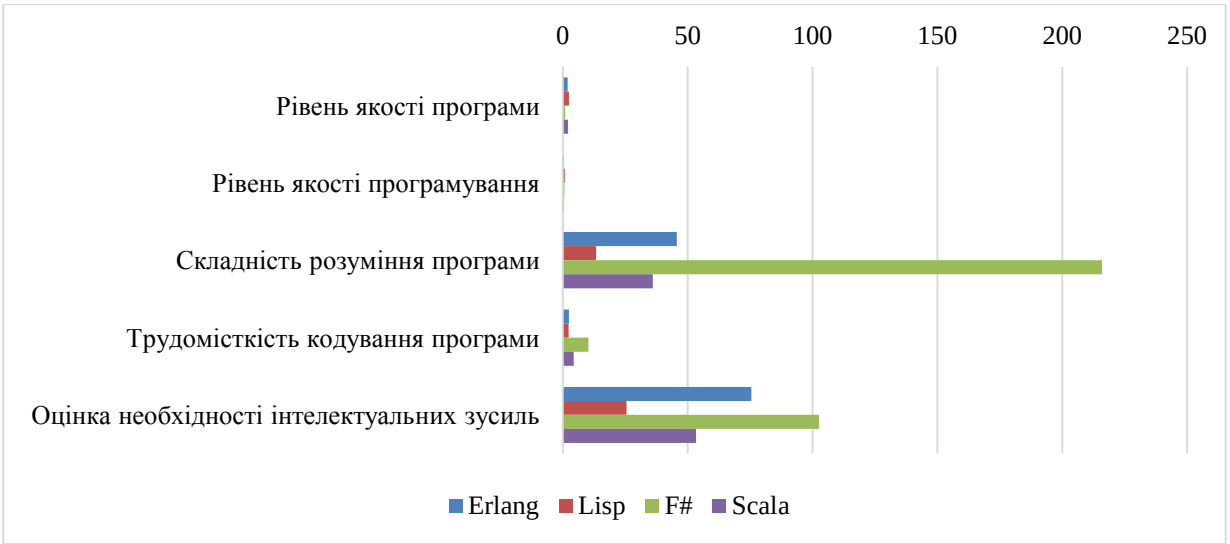

Рис. 2. Розраховані метрики для програм об’ємом менше 30 рядків коду

Fig. 2. Calculated metrics for programs less than 30 lines of code

doi: https://doi.org/10.15802/stp2019/195581

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2019, № 6 (84)

Із наведених діаграм видно, що найменшим за обсягом є код на мові Lisp. Довжина і словник у нього найменші з усіх, із цього випливає, що значення інтелектуальних зусиль і трудомісткості кодування для нього найменші.

Розглянемо порівняння текстів програм за різними сферами застосування. Як приклади вибрано найбільш схожі програми, тобто на різних мовах було реалізовано подібний функціонал.

Перша сфера – це робота з графікою, (рис. 3, 4). Для порівняння взято приклади для роботи з анімацією і SVG. Кількість рядків коду більша за 80, але менша за 120.

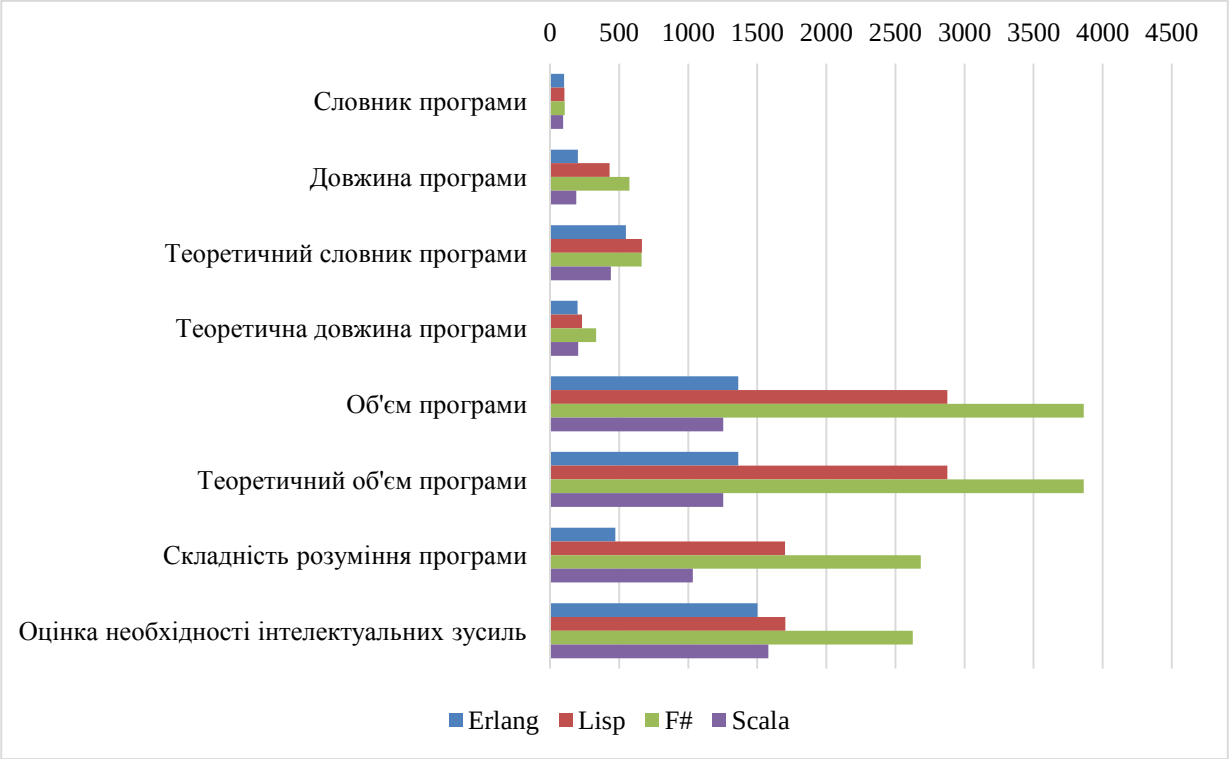

Рис.3. Характеристики програм об’ємом від 80 до 120 рядків коду

Fig. 3. Program features with a volume of 80 to 120 lines of code

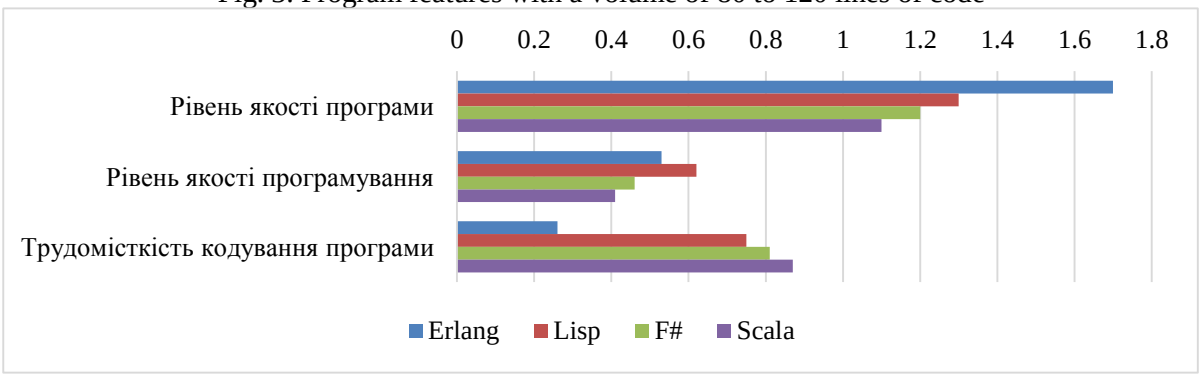

Рис. 4. Розраховані метрики для програм об’ємом від 80 до 120 рядків коду

Fig. 4. Calculated metrics for programs with a volume of 80 and 120 lines of code

Із наведених результатів видно, що словник усіх програм практично однаковий, але за довжиною і за обсягом у явному відриві виступають два приклади – на мові Scala й Erlang. Довжина в них менша порівняно з іншими й майже однакова, якщо порівнювати їх між собою,

doi: https://doi.org/10.15802/stp2019/195581

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2019, № 6 (84)

але за обсягом код Scala трохи менший ніж у Erlang. В одночас код на Scala у 2 рази поступається за складністю розуміння коду на мові Erlang, на це вплинув рівень якості програмування, який у Scala трохи нижчий ніж у Erlang. Код на мові Erlang виглядав більш структуровано, був розбитий на блоки, а код на мові Scala, як уже зазначалося вище, виглядав більш громіздко і мав довгі однорядкові конструкції.

Далі виконано порівняння прикладів веб-програмування. Усі приклади реалізовують роботу веб-сервера. Діаграми представлені на рис. 5 і 6. Кількість рядків коду більше 100, але менше 150.

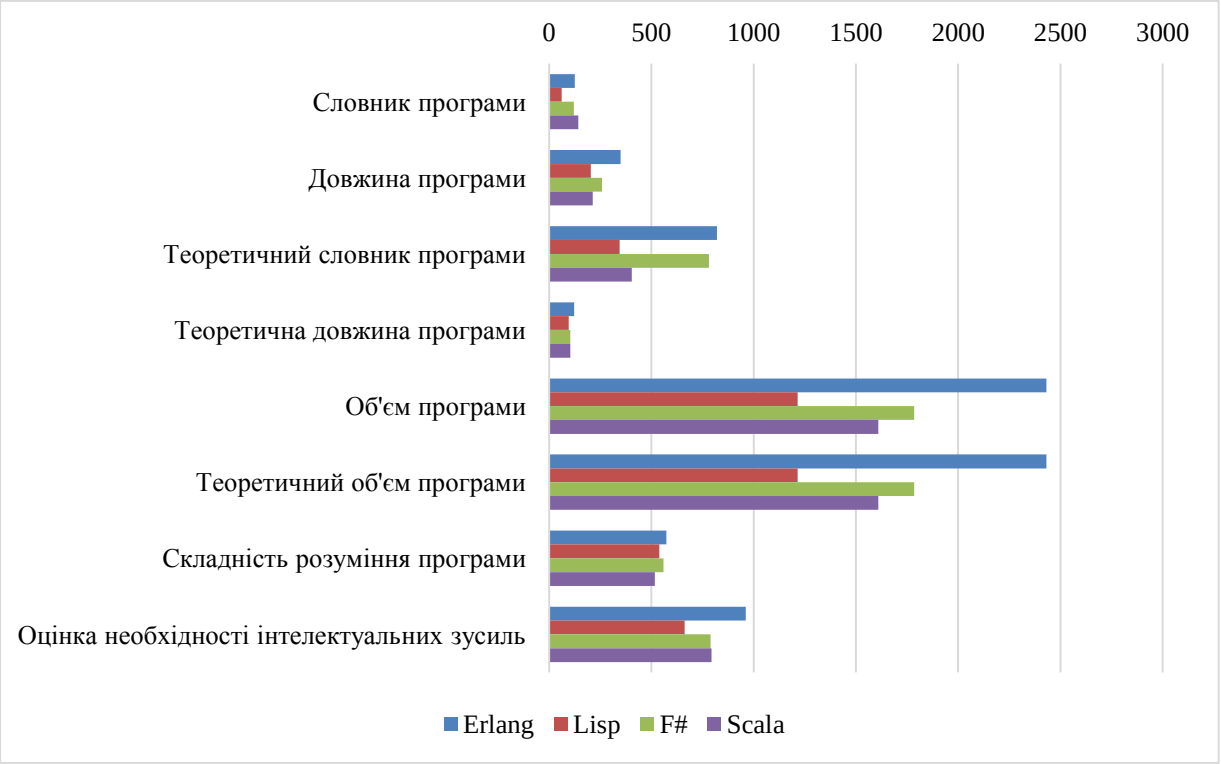

Рис. 5. Характеристики програм об’ємом від 100 до 150 рядків коду

Fig. 5. Features of programs with a volume of 100 to 150 lines of code

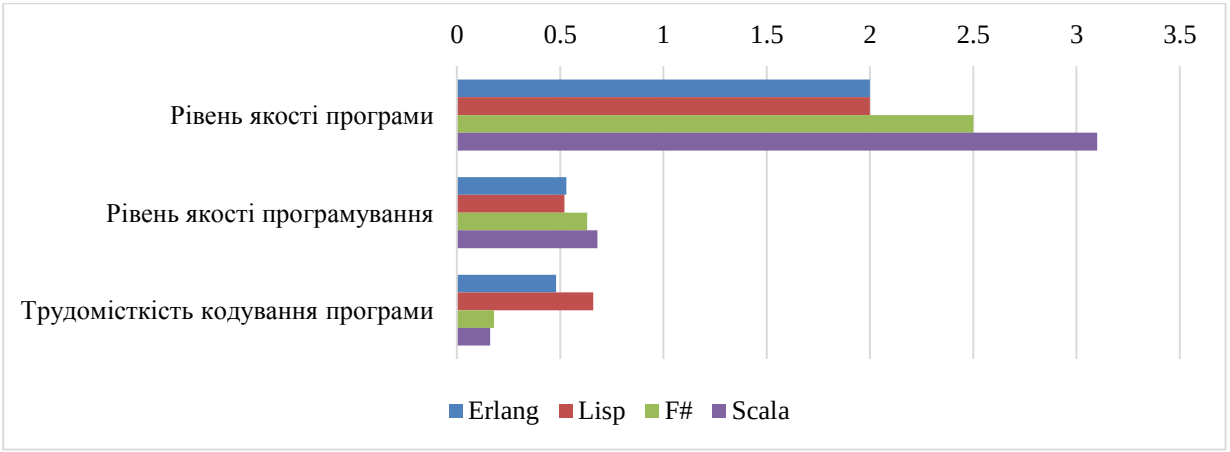

Рис. 6. Розраховані метрики для програм об’ємом від 100 до 150 рядків коду

Fig. 6. Calculated metrics for 100 to 150 lines of code

doi: https://doi.org/10.15802/stp2019/195581

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2019, № 6 (84)

Для реалізації веб-сервера, приклад на мові Lisp налічує найменшу кількість рядків коду. Це видно з діаграм: словник, довжина й обсяг у нього найменші. Але за складністю розуміння приклад на мові Scala трохи виграє у прикладу на мові Lisp, на це вплинув рівень якості програми, який у Scala, виходячи з графіків, вищий ніж у Lisp. Наймісткішим у цьому завданні виявився приклад на мові Erlang, у нього найбільша кількість рядків коду. Хоча словник програми у нього не найбільш об'ємний, але такі показники, як довжина й обсяг у нього найбільші. Разом із низьким показником рівня якості програми це вивело Erlang на останнє місце за складністю її розуміння.

Далі розглянуто реалізацію задач штучного інтелекту, результати наведено на рис. 7 і 8. Для розрахунку метрик використано приклади з реалізацією алгоритмів із книги про штучний інтелект [4]. Кількість рядків коду – від 100 до 150.

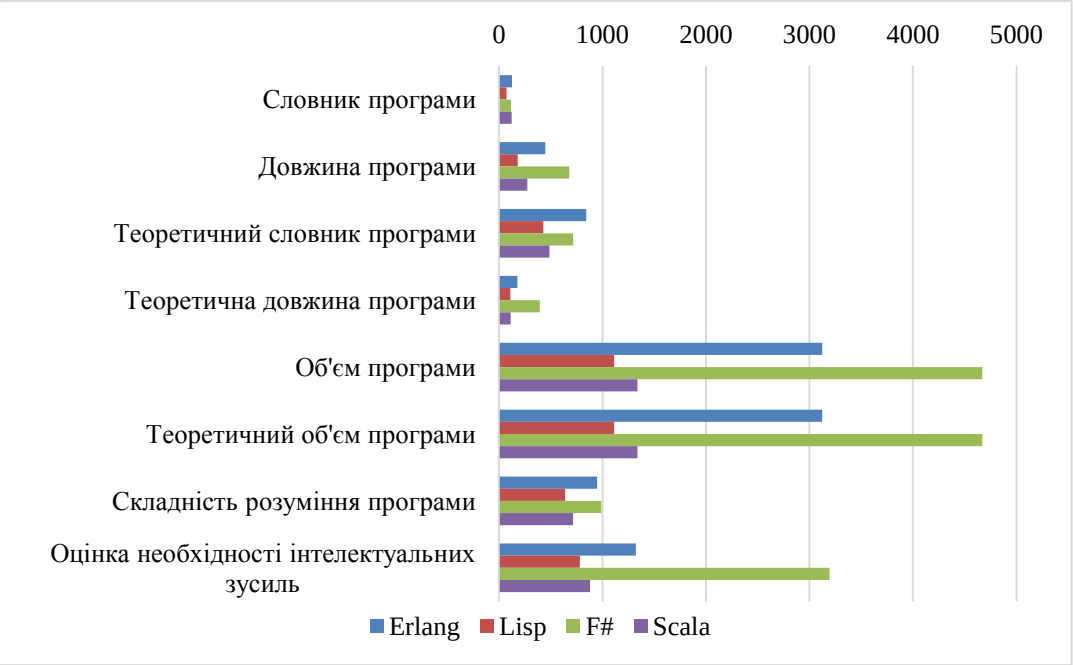

Рис. 7. Характеристики програм об'ємом від 100 до 150 рядків коду

Fig. 7. Features of programs with a volume of 100 to 150 lines of code

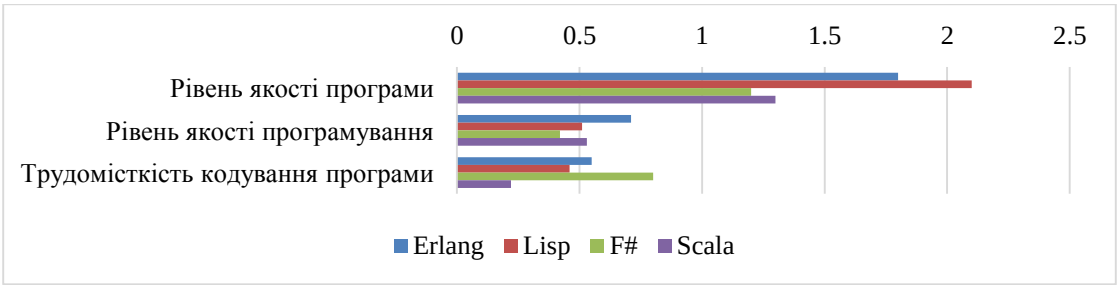

Рис. 8. Діаграма з розрахованими метриками, від 100 до 150 рядків коду

Fig. 8. Diagram with calculated metrics with a volume of 100 to 150 lines of code

Із діаграм видно, що приклади на мовах Lisp та F# сильно відрізняються від інших. Lisp має найменші показники довжини, обсягу та складності розуміння коду. Приклад на Lisp також має найменшу кількість рядків коду. Сам код розбито на невеликі блоки, і він досить читабельний. Найбільші показники виявилися у програмі на мові F#. У ній також найбільша кіль-

doi: https://doi.org/10.15802/stp2019/195581

кість рядків коду. Вона має багато об’ємних конструкцій, які ускладнюють читання коду програми. Крім того, конструкції переповнені характерними для мови F# символами.

Далі розглянуто програми, які використовують роботу з базами даних. Для розрахунку метрик обрані приклади з CRUD – запитами до баз даних, результати наведено на рис. 9 і 10. Кількість рядків коду – від 100 до 120.

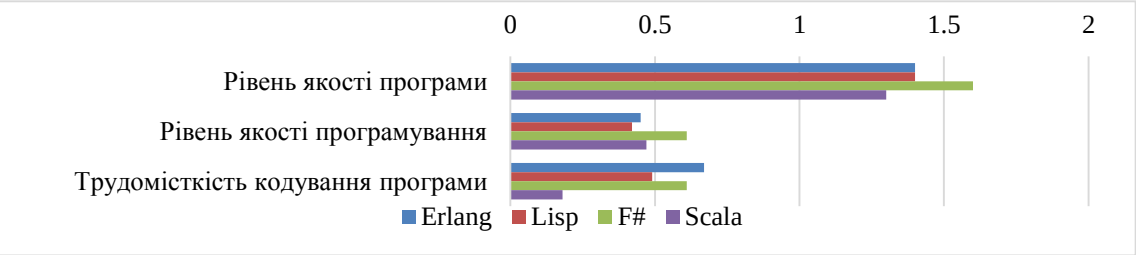

Рис. 9. Характеристики програм об’ємом від 100 до 120 рядків коду

Fig. 9. Features of programs with a volume of 100 to 120 lines of code

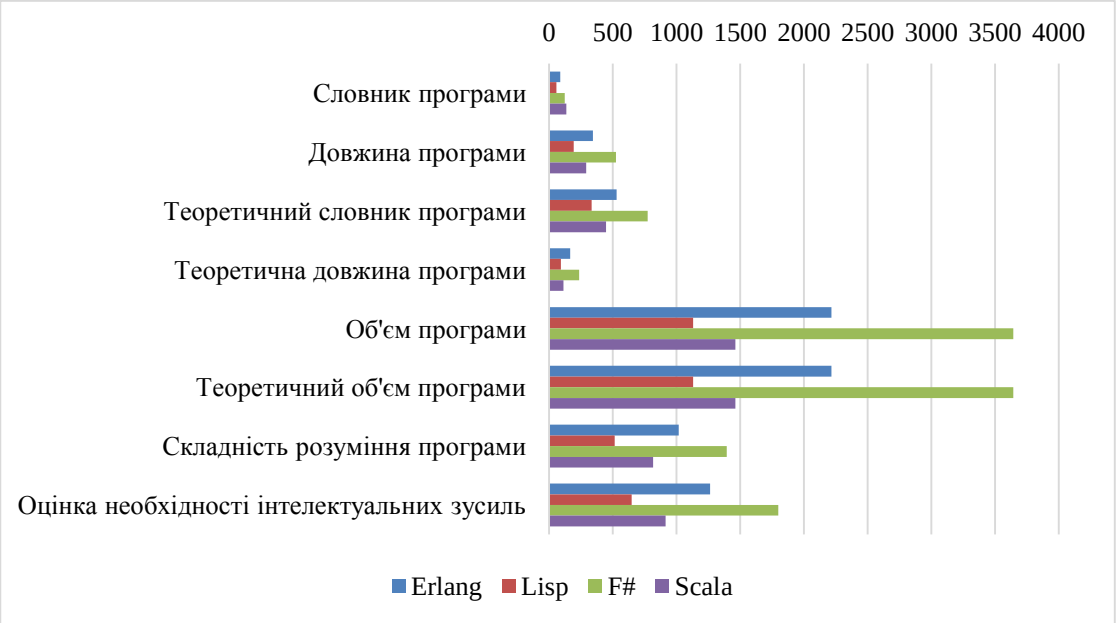

Рис. 10. Розраховані метрики для програм об’ємом від 100 до 120 рядків коду

Fig. 10. Calculated metrics for 100 to 120 lines of code

Приклад на мові F# є найбільш об’ємним. Довжина програми, її об’єм і складність розуміння набагато вищі ніж в інших. Під час аналізу коду з’ясовано, що F# використовує багато громіздких конструкцій порівняно з іншими прикладами, це суттєво знизило читабельність коду. Найменш громіздким виявився приклад на мові Lisp. Майже за всіма показниками він обходить інші приклади. Хоча він і мав кілька об’ємних запитів, але їх було мало, в основному запити були акуратно розбиті на блоки і зручні для читання.

Наукова новизна та практична значимість

У цій роботі вперше представлено методику порівняння функціональних мов програмування за допомогою аналізу прикладів програм, отриманих із відкритих джерел. Розроблено автоматизовану систему, яка виконує розрахунки за допомогою метрик Холстеда.

Отримані висновки та виміри допоможуть під час вибору найбільш ефективної мови фун-

doi: https://doi.org/10.15802/stp2019/195581

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2019, № 6 (84)

кціонального програмування для вирішення конкретних завдань з урахуванням відмінностей у сферах застосування. Розроблене програмне забезпечення дозволяє виконувати виміри для різних текстів програм під час розробки та супроводу складних програмних систем.

Висновки

У роботі розроблено програмне забезпечення, що дозволяє порівнювати приклади програм на різних мовах функціонального програмування. Проведене порівняння дає можливість окреслити такі висновки:

– тексти програм на мові Erlang мають зайву багатослівність, що збільшує об’єм програм і призводить до зниження читабельності коду;

– мова Lisp у більшості прикладів має найменший словник, об’єм і довжину коду. І майже у всіх випадках приклади на мові Lisp налічували найменшу кількість рядків коду, ця програма більш компактна порівняно з іншими мовами програмування;

– мова F#, як і Erlang має проблеми із зайвою багатослівністю записів і великим об’ємом програм, що приводить до зниження читабельності коду;

– мова Scala має досить непогану структуру коду, функції не нагромаджені й загалом зручні для читання. Лише в деяких випадках приклади на мові Scala поступалися за обсягом складністю розуміння коду, написаному на мові Lisp.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Душкин, Р. В. Практика работы на языке Haskell / Р. В. Душкин // Москва : ДМК Пресс, 2016. – 286 с.
2. Одерски, М. Scala. Профессиональное программирование / М. Одерски, Л. Спун, Б. Веннерс. – Питер, 2016. – 688 с.
3. Питер, С. Практическое использование Common Lisp / С. Питер. – Москва : ДМК Пресс, 2017. – 488 с.
4. Рассел, С. Искусственный интеллект. Современный подход. 2-е изд. / С. Рассел, П. Норвиг. – Москва : Вильямс, 2007. – 1410 с.
5. Смит, К. Программирование на F# / К. Смит. – Москва : Символ–Плюс, 2011. – 448 с.
6. Чезарини, Ф. Программирование в Erlang / Ф. Чезарини, С. Томпсон. – Москва : ДМК Пресс, 2015. – 488 с.
7. Dalorzo, E. Functional Programming with Java 8 Functions [Електронний ресурс] / E. Dalorzo. – Режим доступу: https://dzone.com/articles/functional-programming-java-8 – Назва з екрана. – Перевірено : 25.11.2019.
8. GitHub [Електронний ресурс]. – Режим доступу: https://github.com – Назва з екрана. – Перевірено : 25.11.2019.
9. Halstead, Maurice H. Elements of Software Science / Maurice H., Halstead. – New York : North Holland, 1977. – 127 p.
10. Hunt, J. A Beginner’s Guide to Scala, Object Orientation and Functional Programming / J. Hunt. – Springer International Publishing, 2018. – 531 p. doi: 10.1007/978-3-319-75771-1
11. Michaelson, G. An introduction to functional programming through lambda calculus / G. Michaelson. – Dover Publications, 2011. – 336 p.
12. Peyrott, S. Introduction to Immutable.js and Functional Programming Concepts [Електронний ресурс] / S. Peyrott. – Режим доступу: https://auth0.com/blog/intro-to-immutable-js/ – Назва з екрана. – Перевірено : 25.11.2019.
13. Synthesizing functional reactive programs / B. Finkbeiner, F. Klein, R. Piskac, M. Santolucito // Haskell 2019 : Proceedings of the 12th ACM SIGPLAN International Symposium on Haskell. – 2019. – P. 162–175. doi: 10.1145/3331545.3342601
14. Wang, M. Trends in Functional Programming / M. Wang, S. Owens // 18th International Symposium. – Springer, 2017. – 149 p. doi: 10.1007/978-3-319-89719-6

doi: https://doi.org/10.15802/stp2019/195581

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2019, № 6 (84)

И. М. СТОРЧАК[1*], А. П. ИВАНОВ[2*]

[1*]Каф. «Компьютерные информационные технологии», Днипровский национальный университет железнодорожного транспорта имени академика В. Лазаряна, ул. Лазаряна, 2, Днипро, Украина, 49010, тел. +38 (056) 373 15 35, эл. почта storchakigor1@gmail.com, ORCID 0000-0002-8434-9765
[2*]Каф. «Компьютерные информационные технологии», Днипровский национальный университет железнодорожного транспорта имени академика В. Лазаряна, ул. Лазаряна, 2, Днипро, Украина, 49010, тел. +38 (056) 373 15 35, эл. почта iva.dnp@gmail.com, ORCID 0000-0003-1259-6377

АНАЛИЗ МЕХАНИЗМОВ И ЭФФЕКТИВНОСТИ СПЕЦИАЛИЗИРОВАНЫХ ЯЗЫКОВ ФУНКЦИОНАЛЬНОГО ПРОГРАММИРОВАНИЯ

Цель. Авторы ставят целью определить отличия функциональных языков программирования, выявить возможности наиболее популярных языков путём их сравнения и анализа. Для выявления основных возможностей нужно рассмотреть их структуры данных, а также сферы применения. С помощью метрик сложности текстов программ провести анализ и сравнение примеров из различных сфер использования языков. **Методика.** Отобраны пять самых популярных специализированных функциональных языков программирования: Erlang, Lisp, F #, Scala и Haskel. Для получения информации о возможностях каждого из языков изучены их структуры данных, а также сферы применения, проведен обзор официальной документации. Экспериментальная база исследования сформирована из текстов существующих программных систем, полученных из открытого источника и подобранных по схожим направлениям применения и одинаковым объемам текста. Сравнительный анализ примеров программ выполнен по метрикам Холстеда, которые рассчитывают с помощью специально разработанного программного обеспечения. Анализ полученных оценок качества выполнен графическим способом. **Результаты.** Разработано программное обеспечение, которое позволяет получить метрики Холстеда, для входных текстов программ на таких языках функционального программирования, как Erlang, Lisp, F # и Scala. Сложность синтаксиса языка программирования Haskel не позволила использовать метрики для оценки текста, поэтому было проведено только рассмотрение возможностей по документации. С помощью сравнительного анализа показано различие языков и очерчены сферы их использования. Выполнено сравнение примеров разного объема из таких сфер использования, как задачи системного программирования, работа с графикой, математические расчеты, системы искусственного интеллекта, веб-программирование и т. п. **Научная новизна.** Авторы впервые провели сравнительный анализ специализированных языков с помощью метрик сложности текстов, который позволил установить, что язык Lisp имеет самый меньший словарь и длину кода, текст на Scala имеет наиболее структурированный вид, а F # и Erlang отмечаются излишней многословностью. **Практическая значимость.** Полученные выводы и измерения помогут при выборе наиболее эффективного языка функционального программирования для решения конкретных задач с учётом различий в сферах применения. Разработанное программное обеспечение позволяет выполнять измерения для разных текстов программ при разработке и сопровождения сложных программных систем.

Ключевые слова: функциональное программирование; метрики Холстеда; возможности языков; сравнение языков; специализированные функциональные языки; Erlang; Haskel; Lisp; F#; Scala

I. M. STORCHAK[1*], O. P. IVANOV[2*]

1*Dep. «Computer Information Technology», Dnipro National University of Railway Transport named after Academician V. Lazaryan, Lazaryana St., 2, Dnipro, Ukraine, 49010, tel. +38 (098) 971 29 48, e-mail storchakigor1@gmail.com, ORCID 0000-0002-8434-9765
2*Dep. «Computer Information Technology», Dnipro National University of Railway Transport named after Academician V. Lazaryan, Lazaryana St., 2, Dnipro, Ukraine, 49010, tel. +38 (098) 971 29 48, e-mail iva.dnp@gmail.com, ORCID 0000-0003-1259-6377

doi: https://doi.org/10.15802/stp2019/195581

ISSN 2307–3489 (Print), ISSN 2307–6666 (Online)

Наука та прогрес транспорту. Вісник Дніпропетровського національного університету залізничного транспорту, 2019, № 6 (84)

ANALYSIS OF MECHANISMS AND EFFICIENCY OF SPECIALIZED LANGUAGES OF FUNCTIONAL PROGRAMMING

Purpose. The authors aim to determine the differences between functional programming languages, to identify the capabilities of the most popular languages by comparing and analyzing them. To identify the main features, it is necessary to consider their data structures, as well as the application scope. The authors also aim to analyze and compare examples from various fields of language application using metrics of the program texts complexity. **Methodology**. The five most popular specialized functional languages are selected: Erlang, Lisp, F #, Scala and Haskel. An overview of the official documentation was conducted to obtain information on the capabilities of each language; their data structures and the application scope were studied. The experimental research base is formed from texts of the existing open source software systems and matched by similar applications and equal volume of text. Comparative analysis of sample programs is performed using Halsted metrics, which are calculated using specially designed software. The analysis of the received quality assessments is done graphically. **Findings**. Software has been developed to obtain Halsted metrics for program input texts in functional programming languages such as Erlang, Lisp, F # and Scala. The complexity of the Haskel programming language syntax did not allow the use of metrics to evaluate the text, so only a documentation review was performed. Benchmarking shows the differences between languages and outlines their use. The examples of different volumes from such areas of application as system programming tasks, graphing, mathematical calculations, AI systems, web programming, etc. were compared. **Originality.** The authors first conducted a comparative analysis of specialized languages using text complexity metrics, which made it possible to establish that Lisp has the smallest vocabulary and code length, Scala text has the most structured form, and F # and Erlang are marked with extra verbosity. **Practical value.** The findings and measurements will help in selecting the most effective functional programming language for solving specific problems, taking into account differences in applications. The developed software allows making measurements for various program texts when developing and maintaining complex software systems.

Keywords: functional programming; Halstead metrics; language capabilities; language comparison; specialized functional languages; Erlang Haskel Lisp; F #; Scala

REFERENCES

1. Dushkin, R.V. *Practice works in Haskell*. (2016). Moscow: DMK Press. (in Russian)
2. Odersky, M., Spoon, L., & Venners, B. (2011). *Programming in Scala*. Piter. (in Russian)
3. Piter, S. *Practical Common Lisp*. (2017). Moscow: DMK Press. (in Russian)
4. Russell, S., & Norvig, P. (2007). *Artificial intelligence. The modern approach. 2nd Edition.* Moscow: Williams. (in Russian)
5. Smit, K. *Programmirovanie na F#*. (2011). Moscow: Simvol–Plyus. (in Russian)
6. Cesarini, F., & Thompson, S. *Programming in Erlang*. Moscow: DMK Press. (in Russian)
7. Dalorzo, E. Functional Programming with Java 8 Functions. Retrieved from https://dzone.com/articles/functional-programming-java-8
8. GitHub. Retrieved from https://github.com. (in English)
9. Halstead, Maurice H. (1977). *Elements of Software Science*. New York: North Holland. (in English)
10. Hunt, J. (2018). *A Beginner's Guide to Scala, Object Orientation and Functional Programming*. Springer International Publishing. doi: 10.1007/978-3-319-75771-1 (in English)
11. Michaelson, G. (2011). *An introduction to functional programming through lambda calculus*. Courier Corporation. (in English)
12. Peyrott, S. (2016). Introduction to Immutable.js and Functional Programming Concepts. Retrieved from https://auth0.com/blog/intro-to-immutable-js/ (in English)
13. Synthesizing functional reactive programs / B. Finkbeiner, F. Klein, R. Piskac, M. Santolucito // Haskell 2019 : Proceedings of the 12th ACM SIGPLAN International Symposium on Haskell. – 2019. – P. 162–175. doi: 10.1145/3331545.3342601(in English)
14. Wang, M. & Owens, S. Trends in Functional Programming. *18th International Symposium*, 149. doi: 10.1007/978-3-319-89719-6

Надійшла до редколегії: 02.08.2019
Прийнята до друку: 11.11.2019

doi: https://doi.org/10.15802/stp2019/195581