УДК 004.41

ДВІ «РУШНИЦІ» ФУНКЦІОНАЛЬНОГО ПРОГРАМУВАННЯ

Ларіонов О.О., Чухран І.Д.
Науковий керівник – стар. викл. Гребенюк В.О.
Харківський національний університет радіоелектроніки, каф.ШІ
м. Харків, Україна
тел.: +38(066) 588-93-23, +38(099) 382-07-61
e-mail: oleksii.larionov@nure.ua, ivan.chukhran@nure.ua

Comparative analysis of functional and imperative programming paradigms, their learning curves, benefits, and scalability are discussed. Functional programming offers a concise, expressive approach to data and concurrency, is great for mathematical computations, and is easier to refactor and test. Learning functional programming is suggested as it may become increasingly important for developers, making it a valuable area of study to remain competitive in the tech industry.

Функціональне програмування (ФП) – одна з основних парадигм програмування, що виникла у 1920-х роках. Вона базується на концепції функцій, теорії комбінаторної логіки М. Шейнфінкеля та лямбда-численням А. Черча. ФП вперше була реалізована в мові Lisp (1958), розробленій Д. Маккарті, та її діалектах. У 1978 році Дж. Бекусом були виділені важливі концепції чистих функцій, незмінності та референтної прозорості, які презентують ідею “двох рушниць ФП”. На відміну від імперативного підходу, який більшість програмістів обирають на початку розвитку через легкість та доступність, а також послідовний та зрозумілий код, ФП вимагає розвиненого розуміння математичної філософії та потребує кращого розуміння високих абстракцій. Популярність ФП здебільшого виникає серед науковців та людей, які розуміють переваги та недоліки імперативної парадигми.

Робота з даними є важливою частиною розробки програмного забезпечення. У імперативній парадигмі використовуються змінні дані та цикли для операцій з даними, у функціональній – незмінні дані та функції. Функції у функціональній парадигмі є іммутабельними та керуються рекурсивними та умовними виразами [1]. Такий підхід дозволяє використовувати математичні засоби для обробки даних та спрощує процес розробки програмного забезпечення [2]. Імперативні мови програмування керуються змінними та присвоєнням значень до змінних, що робить їх ближчими до архітектури комп'ютерів фон Неймана та швидшими в роботі, але можуть викликати проблеми зі зміною стану об’єктів та помилками програм.

Рефакторинг та тестування – важливий етап розробки програмного продукту, що дозволяє перевірити його відповідність вимогам та виявити проблеми для їх виправлення. Більшість сучасних проектів вимагають

складних архітектурних рішень з використанням шаблонів програмування, що часто ускладнює процес тестування через розміри кодової бази. Стислість та чіткість математичного підходу дозволяють спростити цей процес завдяки функціональному дизайну, який разом з вже описаними принципами ФП дозволяє обирати оптимальну тактику написання коду.

Функціональне програмування є більш ефективним для паралельного програмування, воно дозволяє проектувати програми у вигляді графів, що спрощує розподіл задач між декількома процесами. Відмінність функціональної парадигми від імперативної полягає можливості динамічно розділяти програми на незалежні частини та виконувати їх паралельно. Імутабельність даних зменшує необхідність синхронізації та забезпечує передбачувані результати обчислень. ФП також дозволяє будувати потужні акторні системи з високим рівнем абстракції для розподіленої обробки інформації за допомогою простого функціонального інтерфейсу.

Розширюваність програмного продукту та його підтримка є важливими для будь-якої компанії. Функціональна парадигма забезпечує більшу незалежність для програми та безболісне розширення завдяки декомпозиції та модульності функцій. З іншого боку, об'єктно-орієнтований підхід, оперуючи класами, які є залежними один від одного через наслідування та поліморфізм, ускладнює можливість розширення програмного продукту. Приховані стани, а також складна семантика стану об'єкта у ООП збільшують кількість коду та ускладнюють підтримку програми. Функціональна парадигма використовує функції та їхні композиції, що робить їх універсальними, не має складних правил переходу та прихованих станів, не користується змінними, що дозволяє досягти більш ефективної розробки програмного продукту з мінімальним обсягом коду, безпечністю та більш простою підтримкою.

Завдяки якісному математичному апарату та продуманій концепції, функціональна парадигма дозволяє конкурувати з більш відомим ООП, а “дві рушниці ФП” забезпечують якість та лаконічність програм, безпечність, модульність та розширюваність. Функціональна парадигма є перспективним та серйозним конкурентом ООП, що дозволяє їй бути використаною в комерційних цілях. Функціональна парадигма дозволяє подивитись на звичні проблеми програмування під іншим кутом і знайти оптимальне рішення.

Список використаних джерел:

1. Sebesta, R. W. (2004). Concepts of programming languages (6-те вид.). Pearson/Addison-Wesley.

2. Backus, J. (б. д.). Can programming be liberated from the von Neumann style?: A functional style and its algebra of programs. У ACM Turing award lectures (с. 1977). Association of Computing Machinery. https://doi.org/10.1145/1283920.1283933.