VII Всеукраїнська науково-технічна конференція «Комп’ютерні технології: інновації, проблеми, рішення». 02–03 грудня 2024 р. м. Житомир

*Рихальський Олексій Юрійович,  
ст. викладач "Європейський Університет",  
ст. викладач "DAN.IT Education".*

**Аналіз методів підвищення коректності програмного забезпечення:   
сучасний функціонал компіляторів та інші підходи   
на прикладі мов програмування Scala, Haskell, Rust**

**Метою дослідження** є аналіз методів підвищення коректності програмного забезпечення та їх структурування на прикладі сучасних мов програмування, налаштування компіляторів та інших підходів. Зокрема, дослідження буде зосереджене на використанні функціональних мов програмування Scala, Haskell та Rust, які відомі своєю здатністю забезпечувати високу коректність та надійність ПЗ завдяки своїй експресивності та лаконічності. Також метою є структуризація підходів та формалізація кращих практик програмування на основі 30-річного досвіду здобувача в програмуванні та 10-річного досвіду викладання програмування.

**Актуальність дослідження.** Коректність програмного забезпечення є однією з ключових вимог в сучасному програмуванні, оскільки вона безпосередньо впливає на надійність, безпеку та ефективність використання ПЗ. Зі збільшенням складності програмних систем та критичністю їхнього функціонування, виникає потреба в нових підходах до забезпечення коректності. Разом зі збільшенням складності росте складність покриття тестами, збільшується вірогідність пропустити якийсь крайовий випадок та отримати помилку в реальному застосуванні. Вищезазначені мови програмування надають широкі можливості для статичного аналізу коду, гарантій типів, автоматичного доказу та інших методів, які можуть суттєво підвищити коректність програмного забезпечення. Ці мови все частіше використовуються у вітчизняних та зарубіжних проектах, що підкреслює необхідність детального вивчення та узагальнення їхнього застосування для підвищення коректності ПЗ.

**Об’єктом дослідження** є створення та супроводження сучасного складного ПЗ на всіх стадіях його життєвого циклу.

**Предметом дослідження** є аналіз найтиповіших помилок, аналіз сучасного функціоналу компіляторів, пошук, аналіз і структурування інших підходів до забезпечення коректності програмного забезпечення на усіх стадіях його проектування та реалізації, особливо на початкових стадіях, навіть ще до стадії тестування.

Дослідження намагається розв’язати **наукову проблему**, пов’язану з відсутністю узагальненої методології застосування сучасних мов програмування, можливостей компіляторів та структурування знань на базі накопиченого досвіду для забезпечення коректності ПЗ. Однією з проблем є той факт, що в сьогоднішньому світі задачі потрібно вирішувати швидко, і фактору коректності приділяється не достатньо уваги. Тестування на фінішних стадіях розробки не спроможне покрити всі можливі варіанти, що призводить до помилок, вразливостей та недоліків у системах в реальному використанні, які коштують дуже дорого. Складність ПЗ зростає з кожним днем, єдиним варіантом вирішення проблеми виглядає структурування знань розробників та створення методології, яка може бути використана як повністю так і частково.

**Поточний стан розробки**. У вітчизняній та зарубіжній літературі існує значний обсяг досліджень, присвячених канонічному використанню можливостей мов програмування для підвищення якості, коректності ПЗ, існують безліч «кращих практик», але ніщо не стоїть на місці. Деякі кращі практики згодом перестають бути «кращими», з’являються нові. Все більше публікацій віддають перевагу функціональним мовам програмування, статичній типізації та якісним компіляторам, які розробляються великою кількістю розробників по моделі «Open Source» протягом довгих років тим самим привносячи у мову програмування новий функціонал, за допомогою якого сьогодні можливо вирішити проблему, яка ще вчора була дуже складною, та потребувала значного ресурсу для її реалізації. Також не можна сперечатися з фактом, що більшість ПЗ створюється у стислі строки з використанням вже існуючих бібліотек та фреймворків. В більшості ситуацій ми приймаємо на віру факт «коректності» існуючого ПЗ (бібліотеки, фреймворку) хоча насправді ситуація дуже далека від ідеалу.

Мова програмування **Scala** вже існує і розробляється протягом 20 років. Розроблена для роботи з об’єктно-орієнтованими та функціональними підходами, стала ключовим інструментом для роботи з даними завдяки своїй коректності. Сучасне використання Scala значною мірою зосереджено на розробці масштабованих, надійних та ефективних систем, а її система типів та інші можливості відіграють важливу роль у досягненні цих цілей. Статична типізація дозволяє виявляти помилки ще на етапі компіляції, зменшуючи кількість помилок у рантаймі. Це досягається завдяки підтримці типових параметрів, анотацій типів, та передових можливостей системи типів, таких як залежні типи та підстановлювальні типи, за допомогою яких розробники можуть точно моделювати структуру даних та забезпечувати контрактну поведінку компонентів системи. Scala надає сильну підтримку функціональному програмуванню, яке заохочує розробників використовувати імутабельні структури даних та чисті функції. Це сприяє підвищенню коректності ПЗ, оскільки усуває клас помилок, пов'язаних зі зміною стану, що робить код більш передбачуваним і легшим для тестування. Також важливим моментом є «безшовна» інтеграція зі всіма JVM мовами програмування.

**Haskell** вже існує і розробляється протягом 35 років. Активно використовується в сучасній розробці програмного забезпечення, є однією з найпотужніших та найвиразніших. В першу чергу, це стає можливим завдяки особливостям системи типів та функціональному підходу до програмування. Система типів настільки виразна, що може використовуватися для створення «доказів» правильності програм за допомогою типів. Це робить Haskell надзвичайно підходящою мовою для розробки критично важливих систем, де коректність ПЗ є пріоритетом. Також мова ідеально підходить для вираження концепцій з теорії категорій завдяки підтримці функцій вищого порядку. Використання категорійних абстракцій, таких як монади, функтори, аплікативи, продукти та копродукти, надає розробникам потужні засоби для моделювання обчислень з побічними ефектами, керуванням паралельною обробкою даних та організацією логіки програм у чистий та математично обґрунтований спосіб. Однією з ключових особливостей є система роботи з ефектами за допомогою типів. Монади IO, Maybe, Either надають необхідні засоби для контролю та забезпечують більш передбачуваний та безпечний код, оскільки всі побічні ефекти явно описуються на рівні типів.

**Rust**, відносно молода мова, існує і розробляється вже біля 10 років. Почала розроблятися як мова програмування яка візьме все краще від Scala і Haskell, але на додачу буде більш продуктивнішою завдяки відсутності збирача сміття. Одна з найсучасніших та найперспективніших мов, яка поєднує у собі потужну систему типів, та продуктивність, активно використовується у сферах таких як системне програмування, розробка ігор, та вбудовані системи де важливі коректність та висока швидкість. Статична типізація запобігає багатьом типам помилок, пов’язаних із неправильним використанням типів даних. Rust надає можливість писати швидкий код без необхідності поступатись безпекою або коректністю програмного забезпечення. Компілятор здійснює численні перевірки та оптимізації під час компіляції коду, що дозволяє генерувати ефективний машинний код, який по швидкості можна порівнювати з кодом, написаним на C++.

**Практичні підходи до вирішення проблеми коректності ПЗ** варіюються від використання автоматизованих тестів та статичного аналізу до формальних методів верифікації. Від специфічних практик до жорстких налаштувань компіляторі. У сучасних мовах програмування такі підходи включають:

* Відділення даних від логіки
* Представлення бізнес-сутностей в категоріях продукт / копродукт
* Використання звуження типів (домени), які мають базовий тип, та код додаткової перевірки обмежень
* Іменовані параметри, та заборона мати 2 і більше однакових типи в параметрах функції
* Відділення помилок які ми опрацьовуємо від тих які ми не опрацьовуємо
* Не використання null, використання контекстних обгорток Future, Option, Either, IO
* Максимальне звуження контексту (для зменшення вірогідності помилки)
* Відділення тестів від генерації тестових даних
* Формальні методи верифікації, що забезпечують математичне доведення коректності програми щодо її специфікацій.
* Застосування сучасних IDE та компіляторів.

**Для досягнення цілей дослідження планується використовувати такі методи:**

* 30-річний досвід здобувача в програмуванні дозволяє з певним рівнем окреслити та структурувати ці шаблони, які насправді є «анти-шаблонами», які зустрічалися протягом цього періоду.
* 10-річний досвід здобувача в викладанні програмування дозволяє з певним рівнем окреслити та структурувати шаблони помилок, які є у студентів, які навчаються програмуванню.
* Аналіз питань на stackoverflow.com, оскільки дуже велика кількість питань пов’язана з існуванням цих анти-шаблонів в існуючих бібліотеках та фреймворках і як результат їх неправильного використання.
* Після структурування, кожній групі проблемі буде запропоновано рішення, яке унеможливить або мінімізує повторювання цієї проблеми у майбутньому.
* Запропоновані рішення будуть базуватися на сучасних наукових публікаціях та дослідженнях, присвячених коректності ПЗ.
* Рішення також візьмуть до уваги матеріали профільних конференцій за останні 5-10 років.

**Результати цього дослідження** мають значний потенціал для застосування в різних галузях ІТ-індустрії, де коректність програмного забезпечення є критичною. Зокрема, вони можуть бути корисними у таких сферах:

* **Впровадження безпечних та надійних підходів до розробки програмного забезпечення.** Використання запропонованих методів дозволить забезпечити високу якість та надійність програмного забезпечення, що є важливим для багатьох галузей.
* **Розробка критично важливих систем.** Це стосується банківського сектору, систем керування, телекомунікаційних мереж та інших сфер, де коректність програмного забезпечення є ключовим фактором для забезпечення безперебійної роботи.
* **Використання в освітніх програмах.** Запровадження досліджених методів у навчальних програмах допоможе підготувати програмістів, які спеціалізуються на розробці функціональних та безпечних програмних систем.
* **Автоматизація процесів виявлення та виправлення помилок.** Розроблені інструменти та методології сприятимуть автоматизації процесів виявлення та виправлення помилок, що знизить ризик їх виникнення в коді.
* **Підвищення якості та надійності програмного забезпечення.** Покращення якості ПЗ призведе до зниження витрат на його розробку та підтримку, що, в свою чергу, підвищить задоволеність користувачів і споживачів.

Таким чином, результати цього дослідження мають значний потенціал для покращення процесу розробки, підвищення коректності, безпеки та надійності програмного забезпечення, що дозволить розробникам та компаніям ефективніше відповідати сучасним вимогам ринку.