Рихальський О.Ю., ст. викладач
ПВНЗ «Європейський університет»

МЕТОДИ ПІДВИЩЕННЯ КОРЕКТНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ: СУЧАСНИЙ ФУНКЦІОНАЛ КОМПІЛЯТОРІВ ТА ІНШІ ПІДХОДИ

Коректність програмного забезпечення є не менш важливим за швидкість його розробки. Сучасні системи мають надавати механізми перевірки вхідних параметрів, щоб запобігти некоректним обчисленням та помилкам у роботі. У вихідних параметрах необхідно передбачити можливість відображення ситуацій, коли результат неможливо отримати, із чітким зазначенням причини проблеми, що дозволить потім відфільтрувати «неправильні» результати. Це дозволить користувачам системи своєчасно виявляти та вирішувати помилки. Крім того, варто враховувати питання надійності та масштабованості для забезпечення коректності роботи систем у складних інтеграціях.

Метою роботи є узагальнення та систематизація інформації про проблеми, пов'язані з використанням стандартних типів даних, що є занадто широкими і не дають можливості репрезентувати декілька варіантів результату виконання, зберігаючи при цьому тип даних.

Більшість рішень стосуються сучасних функціональних мов програмування Scala, Haskell та Rust, але майже всі можуть бути використані в інших мовах програмування.

У вітчизняній та зарубіжній літературі існує значний обсяг досліджень, присвячених канонічному використанню можливостей мов програмування для підвищення якості, коректності ПЗ, існують безліч «кращих практик». З часом деякі практики згодом перестають бути «кращими», з'являються нові. Все більше публікацій віддають перевагу функціональним мовам програмування, статичній типізації та якісним компіляторам, щоб зробити пошук помилки максимально швидким. Все більше компіляторів 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 більшість ПЗ створюється у стислі строки з використанням вже існуючих бібліотек. В більшості ситуацій ми приймаємо на віру факт «коректності» існуючого ПЗ хоча насправді ситуація дуже далека від ідеалу.

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.
- Використання звуження типів (*Scala:* бібліотеки *refined, iron*), які мають базовий тип, та код додаткової перевірки обмежень.
- Використання додаткових обгорток над стандартними типами (*Scala: Value Class, Haskell: NewType, Rust: Struct*) щоб надати більше семантики.
- Представлення вхідних/вихідних параметрів бізнес-сутностей в категоріях *enumeration*, таких як *Option[A], Either[E, A], Or[A, B], Result[A, E]*, для явного представлення варіантів та відділення коректних результатів від не коректних.
- Не використання *«null», «0», «-1»* у якості «магічних» значень для репрезентації відсутності результату.
- Використання іменованих параметрів, коли більше ніж один параметр мають однаковий тип.
- Заміна *Boolean* параметрів на відповідні *enumeration* типи, неправильне трактування яких майже неможливо: *Enabled / Disabled, On / Off, Standard / Enhanced.*
- Відділення помилок які ми опрацьовуємо від тих які ми не опрацьовуємо (інфраструктурні).
- Використання можливостей компілятора перевіряти *string* та *integer* літерали в момент компіляції.
- Максимальне звуження контексту (унеможливити доступ до даних, які не потрібні, для зменшення вірогідності помилки)
- Відділення тестів від генерації тестових даних (*Scala: Scalacheck, Haskell: QuickCheck, Rust: proptest*)
- Формальні методи верифікації, що забезпечують математичне доведення коректності програми щодо її специфікацій.

Список використаних джерел

1. Валькман, Ю. Р., Гриценко, Ст. І., Рихальський, А. Ю. Модельно-параметричний простір: теорія та застосування. - Київ: Наук. думка, 2012. - 192 с.

2. Property-Based Testing: Climbing the Stairway to Verification (Artefact) / Z. Chen et al. Zenodo. URL: https://zenodo.org/records/7248640 (date of access: 02.12.2024).