УДК 004.4
Закалюжний М. – ст. гр. СП-41
Тернопільський національний технічний університет імені Івана Пулюя

ДОСЛІДЖЕННЯ ТА РОЗРОБКА ІНФРАСТРУКТУРНИХ РІШЕНЬ ДЛЯ АВТОМАТИЗАЦІЇ ЖИТТЄВОГО ЦИКЛУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ЗАБЕЗПЕЧЕННЯ НАДІЙНОСТІ КОДУ З ВИКОРИСТАННЯМ МОВИ ПРОГРАМУВАННЯ PYTHON

Науковий керівник: к.т.н., доцент Цуприк Г. Б.

Zakaliuzhnyi M.
Ternopil Ivan Puluj National Technical University

RESEARCH AND DEVELOPMENT OF INFRASTRUCTURE SOLUTIONS FOR SOFTWARE LIFE CYCLE AUTOMATION AND CODE RELIABILITY USING THE PYTHON PROGRAMMING LANGUAGE

Supervisor: PhD, Associate Professor H. B. Tsupryk

Ключові слова: об'єктно-орієнтоване програмування, розробка та тестування, життєвий цикл програмного забезпечення.
Keywords: object-oriented programming, development and testing, software development life cycle.

Створення надійних систем безперервної інтеграції та розгортання (CI/CD) потребує гнучкої та продуманої архітектури. Саме тому ідеальним вибором для розробки таких інструментів автоматизації виступає Python. Завдяки лаконічному синтаксису, багатій екосистемі сторонніх бібліотек та потужним вбудованим засобам взаємодії з операційною системою, ця мова дозволяє легко масштабувати проєкт і підтримувати його кодову базу у належному стані.

Фундаментом будь-якого успішного продукту є контроль над усіма етапами його життєвого циклу. У контексті автоматизації це означає, що кожна зміна у коді повинна проходити через етап безперервної перевірки якості ще до моменту формування релізної версії.

Щоб інфраструктурний інструмент був максимально надійним, його архітектура має спиратися на парадигму об’єктно-орієнтованого програмування. Завдяки абстракціям та модульності відбувається чіткий розподіл обов'язків: загальне управління потоком виконання повністю ізолюється від логіки конкретних кроків. Замість монолітної структури формуються автономні підсистеми, кожна з яких відповідає за свою ділянку роботи — чи то обробка конфігураційних файлів, запуск статичних аналізаторів коду (лінтерів), керування збіркою, чи етапи розгортання.

Серцем взаємодії всередині конвеєра виступає спеціалізований диспетчер завдань (task manager). Його роль полягає у реагуванні на зовнішні події (як-от злиття гілок коду або створення pull request), запуску необхідних сценаріїв та контролі за зміною фаз пайплайну. Інтеграція сучасних патернів проєктування гарантує, що середовища залишатимуться ізольованими, а самі завдання зможуть формуватися динамічно і виконуватися паралельно, забезпечуючи високу швидкодію системи.

Критично важливим є принцип взаємодії між вузлами системи автоматизації. Щоб уникнути жорсткої прив'язки між модулями, архітектура проєктується з урахуванням інверсії залежностей та слабкої зв’язності. На практиці це означає, що будь-який скрипт або аналізатор можна модифікувати чи замінити абсолютно безпечно, не ризикуючи порушити працездатність ядра.

Оскільки вимоги до розробки програмного забезпечення постійно еволюціонують, система повинна легко адаптуватися до нових викликів. Базові принципи ООП відкривають шлях до безболісного масштабування: нові типи перевірок, тестові фреймворки або платформи для розгортання можуть бути інтегровані без глибокого втручання в уже існуючий код.

Для забезпечення максимальної прозорості робочого процесу впроваджено комплексні механізми логування та генерації звітів. Вони автоматично фіксують результати статичного аналізу, відстежують відсоток покриття коду тестами та зберігають історію всіх збірок. Завдяки цій аналітиці команда розробників може об'єктивно оцінювати свій прогрес та миттєво реагувати на потенційні проблеми.

Безпосереднє тестування цільового коду є ключовим бар'єром якості. Щойно ініціюється нова збірка, система автоматично запускає набір модульних тестів. Це гарантує ретельну валідацію внесених змін, перевірку бізнес-логіки застосунку та унеможливлює потрапляння нестабільного чи дефектного коду до кінцевого продукту.

У підсумку, запропонована архітектурна модель CI/CD-системи не лише знімає з розробників тягар рутинних перевірок, але й гарантує високу якість інфраструктури. Вона перетворює процес постачання коду на надійний, передбачуваний механізм та створює міцний фундамент для будь-яких майбутніх інтеграцій та покращень.

Література:

1. Олянін, Д., Цуприк, Г. (2025) Transformer Neural Networks in Industry 4.0 / Д. Олянін, Г. Цуприк, Т. Говорущенко, О. Багрій-Заяць, І. Андрущак // Computer Information Technologies in Industry 4.0: proceedings of the 3rd International Workshop (CITI-2025), Ternopil, Ukraine, 11–12 June 2025. – Ternopil : Ternopil Ivan Puluj National Technical University, 2025 (Scopus) https://ceur-ws.org/Vol-4057/

2. Tsupryk, H., Olianin, D. (2025). Vydobuvannia danyh z tekstu vykorystovuiuchy transformerni neironni merezhi [Data extraction from text using Transformer Neural Networks]. Information Technology: Computer Science, Software Engineering and Cyber Security, 125–130, DOI: https://doi.org/10.32782/IT/2025-2-13

3. ОЛЯНІН D., & ЦУПРИК Н. (2025). Огляд ролі трансформерних нейроних мереж у видобувані інформації із неструктурованих даних. Measuring and computing devices in technological processes, 82(2), 360–364. https://doi.org/10.31891/2219-9365-2025-82-52

4. Danyal A. (2024). A Complete Guide to the Software Development Life Cycle (SDLC). https://danyalahmaad.medium.com/a-complete-guide-to-the-software-development-life-cycle-sdlc-a7f4d72e5347

5. Michał Piórkowski (2024). Python for Game Dev: Is It a Good or Bad Idea? https://sunscrapers.com/blog/python-for-game-development-is-it-a-good-or-bad-idea/

6. Sharon Hart (2022). Testing Python module-level code and why it smells. https://medium.com/@sharon.dev/testing-python-module-level-code-and-why-it-smells-3bf1cc5a79d5