Гриб П. Ю., здобувач,
Варганова Д. О., ст. викладач,
Окунькова О. О., ст. викладач,
Державний університет «Житомирська політехніка»

ЕФЕКТИВНІСТЬ ВИКОРИСТАННЯ ПРИНЦИПІВ SOLIDI PATTERNS ПРИ НАПИСАННІ «ЧИСТОГО КОДУ»

У сфері програмування сьогодні велике значення приділяється написанню коду, який був би зрозумілим та легким у обслуговуванні. Численні методики написання коду пропонують різні підходи, але лише деякі з них можуть гарантувати високий рівень якості та легкості у підтримці.

«Чистий код» означає не лише функціональний код, але й такий, що є інтуїтивно зрозумілим, простим для змін та підтримки. Створення такого коду вимагає додаткової дисципліни та уважності. Код, який відповідає критеріям чистоти, є менш вразливим до помилок, його легше удосконалювати та розширювати.

Проаналізуємо ефективність застосування принципів SOLID та патернів проєктування.

SOLID являє собою набір базових принципів, які використовуються в об'єктно-орієнтованому програмуванні для створення програм, які є ефективними, масштабованими та легко змінюються. Принципи SOLID та його можливого застосування:

- Принцип єдиної відповідальності (Single Responsibility Principle – SRP). Кожен клас або модуль повинен мати тільки одну причину для зміни. Наприклад, клас, що відповідає за зберігання даних користувача, має відповідати тільки за це і не має займатися іншими функціями, такими як обробка вводу.
- Принцип відкритості/закритості (Open/Closed Principle – OCP). Код повинен бути відкритим для розширення, але закритим для зміни. Наприклад, ми можемо мати базовий клас для зберігання даних, який можна розширювати новими функціями, не змінюючи при цьому сам клас.
- Принцип підстановки Барбари Лісков (Liskov Substitution Principle – LSP). Об'єкти класів-спадкоємців повинні бути здатні замінювати свої батьківські класи без зміни ідеї програми. Наприклад, якщо у нас є клас для обробки платежів, ми можемо замінити його конкретними класами для кредитних карт та PayPal, не змінюючи основної логіки.
- Принцип розділення інтерфейсу (Interface Segregation Principle – ISP). Клієнти не повинні залежати від методів, які вони не використовують. Наприклад, якщо у нас є інтерфейс для зберігання даних користувача, інші класи мають залежати тільки від методів, які їм потрібні для роботи з цими даними.
- Принцип інверсії залежності (DependencyInversionPrinciple – DIP). Залежності повинні бути встановлені на абстракції, а не на конкретних класах. Наприклад, ми можемо використовувати інтерфейси для взаємодії з різними типами баз даних, дозволяючи легко змінювати базу даних без зміни основного коду.

Загальноприйнятим вирішенням типових проблем, що виникають під час проєктування програмного забезпечення є патерни проєктування. Вони є відображенням досвіду та кращих практик розробки програмного забезпечення і надають можливість розробникам використовувати структуровані методики для вирішення конкретних проблем, створювати програмне забезпечення, яке є ефективним та масштабованим. Найвідоміші класифікації за кількома критеріями, такими як ціль використання, рівень абстракції та область застосування: Strategy, Factory, Singleton, Prototype, Observer. Ці патерни використовуються в широкому спектрі областей, включаючи розробку програмного забезпечення, веб-розробку, мобільну розробку та інші. Вони є важливим інструментом для будь-якого програміста, який прагне створювати високоякісне та ефективне програмне забезпечення. Найчастіше є використання патернів з MV-сім'ї, а саме MVC. Шаблон Model-View-Controller є одним із найбільш популярних та широко використовуваних у веб-розробці. Він дозволяє розділити структуру програми на три основні компоненти: Модель, Вид та Контролер. Шаблон MVC дозволяє відокремити логіку програми від представлення та взаємодії з користувачем, що робить програмне забезпечення більш гнучким, легким для розуміння та підтримки.

Отже, принципи SOLID та патерни проектування створюють надійну основу для розробки програмного забезпечення. Використання цих методик забезпечує не лише якість коду, але й прискорює процес розробки, збільшує його функціональність, зменшуючи ймовірність виникнення помилок та сприяє створенню стабільного та довговічного програмного забезпечення.

Список використаних джерел

1. Мартін Р. Чистий код: створення, аналіз, рефакторинг / Роберт Мартін. 2019. – 416 с.
2. Принципи SOLID [Електронний ресурс] – Режим доступу до ресурсу: https://training.epam.ua/ua/blog/602.