СЕКЦІЯ «КОМП'ЮТЕРНІ СИСТЕМИ І МЕРЕЖІ»

УДК 004.42
Дьячук Т.С.
старш. викл. НУ «Запорізька політехніка»

ВІД «КОД ПРАЦЮЄ» ДО «КОД ЯКІСНИЙ»: ФОРМУВАННЯ ІНЖЕНЕРНОГО МИСЛЕННЯ В ПРОЦЕСІ НАВЧАННЯ

При навчанні програмуванню, критично важливо зміщувати акцент із досягнення короткострокової мети – «код виконується» - на довгострокову - "код якісний". Код, який працює – це тільки початок. Якісний код - це той, який легко читати, тестувати, підтримувати й масштабувати.

У навчальному процесі ми спостерігаємо типовий шаблон поведінки студентів, при якому домінує прагнення "запустити код будь-якою ціною". Завдання викладача - поступово формувати інженерне мислення, в основі якого лежать принципи Clean Code, SOLID, грамотне іменування, правильне структурування проєкту та покриття тестами.

Clean Code ("чистий код") - це підхід до написання коду, який легко читати, розуміти й підтримувати. Основні принципи: осмислені імена, малі функції з єдиною відповідальністю, мінімум дублювання, прозора структура, самодокументованість, відсутність "магічних чисел" (числових значень, які жорстко закодовані в коді без пояснення їхнього значення), тестованість і єдиний стиль написання. Чистий код має бути зрозумілим без коментарів і виглядати так, ніби його писала людина, якій не байдуже. Для студентів і молодих розробників принципи Clean Code закладають основу професійного мислення. Вони вчать працювати не лише з кодом, а з продуктом, командою і підтримкою проєкту. Чистий код знижує когнітивне навантаження, спрощує командну роботу та скорочує час розробки в перспективі.

Принципи SOLID містить п'ять правил об'єктно-орієнтованого проєктування, які допомагають писати гнучкий і підтримуваний код. Принцип єдиної відповідальності (S - Single Responsibility Principle) означає, що кожен клас має виконувати лише одну задачу. Принцип відкритості/закритості (O - Open/Closed Principle) закликає розширювати функціональність без зміни існуючого коду. Принцип підстановки Лісков (L - Liskov Substitution Principle) вимагає, щоб підкласи могли замінювати базові класи без порушення логіки. Принцип розділення інтерфейсу (I - Interface Segregation Principle) пропонує створювати вузькі інтерфейси, які не змушують реалізовувати зайве. Принцип інверсії залежностей (D - Dependency Inversion Principle) закликає залежати від абстракцій, а не від конкретних реалізацій. Разом ці принципи формують основу якісної архітектури програмного забезпечення.

Таким чином принципи SOLID формують архітектурне мислення, пояснюють зв'язки між класами та компонуванням проєкту, навчають масштабувати без порушення існуючої логіки та дають студентам інструментарій для професійної розробки. Ці принципи не стільки про ідеальний код, скільки про контроль над зростаючою складністю.

Розглянемо грамотне іменування - ключовий аспект якісного програмування, який безпосередньо впливає на читабельність, зрозумілість і підтримуваність коду. Ім'я повинно чітко відповідати суті: функції мають містити дієслова (наприклад, calculateTotal()), а змінні - іменники (userList, invoice). Потрібно уникати узагальнених назв типу data, info, stuff, а також скорочень, які не є загальноприйнятими. Імена мають бути послідовними, зрозумілими без додаткових коментарів й не залежати від деталей реалізації. Важливо, щоб код "читався як текст", без потреби звертатися до коментарів. Студентів слід навчати думати про назву як про інтерфейс для іншого розробника, або для себе в майбутньому.

Правильне структурування проєкту забезпечує зрозумілість, масштабованість і зручність у підтримці коду. Логіка, дані, інтерфейс і сервіси мають бути чітко відокремлені відповідно до їхніх функцій. Кожен файл і папка повинні мати своє місце і змістовну назву. Важливо уникати хаотичних структур та узагальнених назв на кшталт utils або stuff. Така організація спрощує навігацію, командну роботу, тестування і розвиток проєкту. Це основа інженерного підходу навіть у невеликих завданнях.

Покриття тестами - це частина якісної розробки, яка дозволяє перевіряти працездатність коду автоматично, виявляти помилки на ранніх етапах і впевнено вносити зміни без ризику порушити існуючу логіку. Тести мають охоплювати ключову бізнес-логіку, граничні випадки й типові сценарії використання. Важливо навчати не просто писати тести, а проєктувати код так, щоб його було зручно тестувати: модульно, без прихованих залежностей. Тестування формує інженерне мислення, підвищує довіру до коду і є критерієм зрілості проєкту.

Таким чином студенти ще на ранніх етапах навчання повинні розуміти, що розробник програмного забезпечення - це не просто технічний виконавець, який пише код за вимогами, а інженер, який працює з обмеженнями, командною динамікою та довготривалою підтримкою проєктів. Завдання сучасної освіти в IT - не лише дати інструмент, а й сформувати мислення, орієнтоване на якість, ефективність і відповідальність за код, що створюється. У цьому контексті мета якісної IT-освіти полягає не лише в передачі технічних знань чи навичок володіння мовами програмування та фреймворками. В процесі навчання має формуватися інженерне мислення - здатність бачити систему в цілому, розуміти наслідки архітектурних рішень, дотримуватись принципів якості коду. Такий підхід сприяє підготовці не просто програмістів, а свідомих і зрілих інженерів.