**Лабораторна робота**

**Інструментальні засоби розробки програмного забезпечення**

**Тема:**

Практичне застосування системи контролю версій Git та юніт-тестування у процесі розробки програмного забезпечення.  
Формування повного робочого циклу з GitHub — від створення репозиторію до Pull Request.

**Мета роботи:**

Отримати практичні навички використання сучасних інструментів розробки ПЗ, а саме:

* роботи з системою контролю версій Git та сервісом GitHub;
* побудови історії комітів і гілок проєкту;
* написання юніт-тестів до існуючого коду;
* формування правильного процесу створення Pull Request та командної взаємодії у GitHub.

**Зміст лабораторної роботи**

**1. Підготовка вихідного коду**

Студент обирає код, з яким буде працювати:

* можна використати власний навчальний або лабораторний проєкт;
* можна взяти відкритий код з open-source репозиторію;
* або використати будь-який невеликий приклад (наприклад, консольну програму, утиліту, скрипт тощо).

Далі студент створює на GitHub **новий репозиторій**, завантажує до нього вихідний код і налаштовує базову структуру проєкту.  
Обов’язково має бути створений файл **README.md** із коротким описом призначення програми.  
Початковий стан фіксується у першому коміті з повідомленням на кшталт *“Initial project version”*.

**2. Робота з гілками та історією комітів**

У межах репозиторію студент починає роботу в **новій гілці**, наприклад feature/unit-tests.  
Усі зміни оформлюються окремими комітами, які логічно описують етапи роботи:  
створення тестів, виявлення проблем, виправлення коду тощо.

Коміт-повідомлення повинні бути короткими, змістовними й відповідати змісту змін.  
У процесі роботи студент демонструє правильне користування основними командами Git (add, commit, push, merge, checkout тощо) та підтримує чисту структуру репозиторію.

**3. Написання юніт-тестів**

Основна практична частина роботи полягає у створенні **юніт-тестів** до наявного коду.  
Мета цього етапу — навчитися ізольовано перевіряти роботу окремих функцій або класів.

Студент створює папку для тестів (наприклад, tests) і поступово додає нові тести, що перевіряють:

* базову функціональність програми;
* граничні та виняткові випадки;
* помилки, які були знайдені під час налагодження (debug).

Тести розробляються за допомогою фреймворків, що підтримуються обраною мовою програмування (NUnit, JUnit, unittest, pytest, Jest тощо).

**4. Паралельне ведення Git**

Під час розробки тестів студент одночасно демонструє **культуру роботи з Git**:

* створює логічні гілки під окремі задачі;
* супроводжує зміни інформативними комітами;
* регулярно синхронізує роботу з GitHub;
* підтримує зрозумілу структуру проєкту й опис у README.

Таким чином, у ході лабораторної роботи одночасно відпрацьовується **написання тестів** і **правильне ведення репозиторію GitHub**.

**5. Створення Pull Request**

Після завершення роботи над тестами студент створює **Pull Request (PR)** з гілки, у якій він працював, до головної гілки репозиторію (main або master).

У описі Pull Request потрібно зазначити:

* що саме було додано або змінено;
* які модулі або функції покриті тестами;
* чи були знайдені та виправлені помилки.

Посилання на створений Pull Request надсилається викладачу для перевірки.

**Результати виконання**

Після завершення лабораторної роботи студент повинен надати:

1. Посилання на власний GitHub-репозиторій із початковим кодом.
2. Історію комітів, що демонструє послідовний процес розробки тестів.
3. Опис створених тестів та їх призначення.
4. Посилання на створений Pull Request для перевірки.
5. Короткий опис знайдених та виправлених помилок (якщо такі були).

**Форма звіту**

1. Тема, мета, короткий теоретичний вступ.
2. Посилання на репозиторій GitHub.
3. Скріншоти основних етапів роботи (створення гілки, коміти, Pull Request).
4. Опис процесу написання юніт-тестів.
5. Висновки про набуті навички.