МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Тернопільський національний технічний університет
імені Івана Пулюя

Кафедра програмної інженерії

Методичні вказівки до лабораторних робіт

з дисципліни

«Основи програмної інженерії (SE201)»

CEH ID: **6822**

для здобувачів першого (бакалаврського) рівня вищої освіти
ОПП «Інженерія програмного забезпечення»

Тернопіль — 2025

УДК 004.4(075.8)
ББК 32.973.202я73
Б83

Укладач:
В.М. Бревус, PhD

Рецензент:
В.В. Яцишин, к.т.н., доцент
завідувач кафедри систем штучного інтелекту та аналізу даних ТНТУ ім. І. Пулюя

Методичні вказівки розглянуто й затверджено на засіданні кафедри програмної інженерії протокол № «1» вересня від 2025 року.

Схвалено та рекомендовано до друку на засіданні методичної комісії факультету комп'ютерно-інформаційних систем та програмної інженерії Тернопільського національного технічного університету імені Івана Пулюя протокол № «1» вересня від 2025 року.

Б83 Методичні вказівки до лабораторних робіт з дисципліни «Основи програмної інженерії (SE201)» для здобувачів першого (бакалаврського) рівня вищої освіти ОПП «Інженерія програмного забезпечення» всіх форм навчання / Укладач: Бревус В.М. – Тернопіль: ТНТУ імені Івана Пулюя, 2025 – 33 с.

Відповідальний за випуск: М.Р. Петрик, докт. фіз.-мат. наук, професор

АНОТАЦІЯ

У методичних вказівках представлено комплексний наскрізний проєкт «Розробка MVP чат-платформи з використанням практик програмної інженерії». Проєкт присвячений вирішенню реальних завдань розробки програмного забезпечення з використанням сучасних методологій та інструментів програмної інженерії.

Лабораторні роботи охоплюють весь цикл розробки програмного забезпечення: від аналізу вимог та архітектурного проєктування до реалізації, тестування та розгортання системи. Особливу увагу приділено командній роботі, використанню систем контролю версій, створенню технічної документації та дотриманню професійних стандартів.

Методичні вказівки призначені для здобувачів першого (бакалаврського) рівня вищої освіти, які навчаються за освітньо-професійною програмою «Інженерія програмного забезпечення». Вони також будуть корисними для студентів технічних спеціальностей, що вивчають основи програмної інженерії та розробки програмного забезпечення.

Ключові слова: програмна інженерія, життєвий цикл розробки, командна співпраця, тестування програмного забезпечення, архітектурне проєктування, MVP.

ABSTRACT

The methodical guidelines present a comprehensive end-to-end project "Development of Chat Platform MVP using Software Engineering Practices". The project is dedicated to solving real-world software development challenges using modern software engineering methodologies and tools.

The laboratory works cover the complete software development lifecycle: from requirements analysis and architectural design to implementation, testing, and system deployment. Special attention is given to teamwork, version control systems usage, technical documentation creation, and adherence to professional standards.

The methodical guidelines are intended for students of the first (bachelor's) level of higher education enrolled in the "Software Engineering" educational and professional program. They will also be useful for students of technical specialties studying the fundamentals of software engineering and software development.

Keywords: software engineering, development lifecycle, team collaboration, software testing, architectural design, MVP.

Зміст

Вступ

Деталі щодо оформлення програмного коду та змісту звіту лабораторних робіт узгоджуються з викладачем у системі електронного навчання відповідного курсу. Приклад оформлення звіту з титульним аркушем, структурою та рекомендаціями наведено в додатку А.

Усі лабораторні роботи повинні бути виконані з дотриманням принципів академічної доброчесності, відповідно до Положення про академічну доброчесність учасників освітнього процесу Тернопільського національного технічного університету імені Івана Пулюя.

Наскрізний проєкт

«Розробка MVP чат-платформи з використанням практик програмної інженерії»

Протягом серії лабораторних робіт здобувачі розроблятимуть мінімально життєздатний продукт (MVP) чат-платформи. Проєкт спрямований на практичне засвоєння ключових принципів програмної інженерії через вирішення реальних завдань розробки програмного забезпечення.

Бізнес-обґрунтування

Сучасна програмна індустрія потребує спеціалістів, які володіють не лише технічними навичками, але й розуміють повний цикл розробки програмного забезпечення. Створення чат-платформи дозволяє імітувати роботу в команді розробників над реальним продуктом, від аналізу вимог до розгортання та підтримки. Цей проєкт імітує роботу у стартапі або команді розробки нового продукту, де кожен учасник має зрозуміти свою роль у загальному процесі.

Ключові переваги підходу:

- **Системний підхід:** Розуміння взаємозв'язків між різними етапами розробки програмного забезпечення.
- **Командна робота:** Практичні навички роботи в команді з використанням сучасних інструментів співробітництва.

- **Якість коду:** Застосування практик написання якісного, тестованого та документованого коду.
- **Професійні стандарти:** Знайомство з етичними та професійними стандартами галузі.

Технічний підхід

Проєкт базується на використанні сучасних методологій програмної інженерії та інструментів розробки для створення веб-додатку реального часу.

1. **Систематичний аналіз:** Використання методик збору та аналізу вимог для чіткого розуміння потреб користувачів.
2. **Архітектурне проєктування:** Створення UML-діаграм та технічної документації для планування структури системи.
3. **Ітеративна розробка:** Поетапна реалізація функціоналу з постійним тестуванням та валідацією.
4. **Контроль якості:** Застосування автоматизованого тестування та практик безперервної інтеграції.

Структура лабораторних робіт

ЛР№1: Налаштування середовища розробки та командна співпраця

- Налаштування професійного середовища розробки (VS Code, Git).
- Створення репозиторію проєкту та налаштування командної роботи.
- Вивчення практик управління версіями та розгалуженнями.
- Планування архітектури проєкту та розподіл ролей у команді.

ЛР№2: Аналіз вимог та архітектурне проєктування

- Створення специфікації вимог програмного забезпечення (SRS).
- Розробка User Stories та сценаріїв використання.

- Проєктування архітектури системи з використанням UML-діаграм.
- Планування структури бази даних та API інтерфейсів.

ЛР№3: Реалізація та тестування основного функціоналу

- Імплементація ключових компонентів системи (автентифікація, чат).
- Написання unit-тестів для критичних функцій системи.
- Застосування практик чистого коду та рефакторингу.
- Інтеграція компонентів та проведення інтеграційного тестування.

ЛР№4: Професійні стандарти та кар'єрне планування

- Аналіз етичних стандартів та професійної відповідальності.
- Створення професійного портфоліо та LinkedIn профілю.
- Дослідження ринку праці та кар'єрних можливостей.
- Підготовка презентації проєкту та технічної документації.

Очікувані результати

По завершенні проєкту здобувачі матимуть:

1. **Функціональний MVP** чат-платформи з базовими можливостями.
2. **Глибоке розуміння** життєвого циклу розробки програмного забезпечення.
3. **Практичні навички** роботи з сучасними інструментами розробки та співробітництва.
4. **Досвід командної роботи** над технічним проєктом.
5. **Портфоліо проєкту** для демонстрації професійних навичок.

Зв'язок з теоретичним курсом

Лабораторні роботи тісно інтегровані з лекційним матеріалом:

- **Тема 1-2:** Вступ до програмної інженерії, процеси розробки — ЛР№1.
- **Тема 3-4:** Інженерія вимог, проєктування — ЛР№2.
- **Тема 5-6:** Конструювання, тестування — ЛР№3.
- **Тема 7-8:** Професійна практика, етика — ЛР№4.

Інструментарій та ресурси

Програмне забезпечення: VS Code, Git, GitHub, Node.js, Python (залежно від обраної технології).

Методології: Agile/Scrum принципи, практики DevOps, Test-Driven Development (TDD).

Обчислювальні ресурси: Стандартний ноутбук або комп'ютер з можливістю встановлення середовища розробки.

Лабораторна робота №1: Налаштування професійного середовища розробки

Цілі

Після завершення цієї лабораторної роботи, ви зможете:

- **Застосовувати знання на практиці**: Налаштовувати робоче середовище, включаючи IDE (VS Code) та систему контролю версій (Git).
- **Працювати в команді**: Використовувати Git та GitHub для спільної роботи над проєктом, включаючи створення гілок, комітів, та проведення рецензування коду через Pull Requests.
- **Реалізовувати фази життєвого циклу ПЗ**: Ініціалізувати проєкт, створювати його початкову структуру та документацію, що є першим етапом у життєвому циклі розробки.
- **Застосовувати інструментальні засоби**: Впевнено використовувати базові та просунуті команди Git для керування версіями коду.
- **Оформлювати програмну документацію**: Створювати та підтримувати документацію проєкту у форматі Markdown, включаючи звіти та файли README.
- **Організовувати роботу над проєктом**: Використовувати GitHub Project для планування завдань та комунікації з викладачем.
- **Аналізувати існуючі проєкти**: Досліджувати структуру, гілки та історію змін в існуючому git-репозиторії курсу.

Основна мета — сформувати у вас практичні навички для роботи в команді, закладаючи міцний фундамент для наступних етапів розробки вашого проєкту.

Теоретичне підґрунтя

- **Програмна інженерія та життєвий цикл ПЗ (SDLC):** Розробка ПЗ — це інженерна дисципліна, що охоплює всі етапи від ідеї до підтримки го-

тового продукту. Цей процес називається життєвим циклом ПЗ (Software Development Life Cycle).

- **Система контролю версій Git:** Це розподілена система керування версіями, що дозволяє відстежувати зміни у файлах та координувати роботу кількох розробників.
- **GitHub Organization:** Це спільний робочий простір для команди або установи, що дозволяє централізовано керувати доступом, репозиторіями та учасниками.
- **GitHub Repository:** Це сховище для вашого проєкту, яке містить усі файли, історію їхніх змін та інструменти для спільної роботи.
- **GitHub Project:** Це інструмент для планування та відстеження роботи, подібний до дошки Kanban, що допомагає організувати завдання, планувати ітерації та візуалізувати прогрес.
- **Формат Markdown для документації:** Це легка мова розмітки, що дозволяє створювати форматований текст за допомогою простого синтаксису. В межах курсу він є рекомендованим, оскільки файли `.md` є звичайними текстовими файлами, що легко версіонуються за допомогою Git, добре читаються як у сирому вигляді, так і після рендерингу на платформах типу GitHub. Це підтримує підхід "документація як код" (docs-as-code).
- **Командна робота та процеси розробки:** Ефективна командна робота вимагає узгоджених процесів. Git Flow та GitHub Flow є популярними моделями організації роботи з гілками. Ключовим елементом є *Pull Request* (PR) — запит на включення змін з однієї гілки до іншої, що дозволяє проводити обговорення та рецензування коду (*code review*) перед його інтеграцією.

Рекомендації щодо комітів та роботи з Git

- **Створюйте атомарні коміти.** Кожен коміт має містити одну логічну зміну. Це спрощує рецензування коду, відстеження історії та, за потреби, скасування змін.

- **Пишіть змістовні повідомлення до комітів.** Повідомлення має чітко описувати, що саме було змінено і чому. Рекомендується дотримуватися формату, де перший рядок є коротким заголовком (до 50 символів), а тіло коміту (за потреби) містить детальніший опис.

- **Синхронізуйте зміни регулярно.** Регулярно виконуйте `git pull` з віддаленого репозиторію, щоб мати актуальну версію коду та уникати конфліктів злиття. Надсилайте свої зміни (`git push`) на віддалений репозиторій, щоб поділитися ними з командою.

- **Працюйте у гілках.** Ніколи не працюйте безпосередньо в основній гілці (`main` або `master`). Для кожного нового завдання чи функціоналу створюйте окрему гілку. Це ізолює вашу роботу та спрощує процес інтеграції через Pull Requests.

Що потрібно зробити

Ось покроковий план для успішного виконання лабораторної роботи. Виконуйте завдання послідовно, щоб нічого не пропустити.

1. **Аналіз репозиторію курсу:** Дослідіть структуру каталогів для лабораторної роботи в репозиторії курсу (`SE201-Software-Engineering-Fundamentals`). Ознайомтесь із вмістом каталогу `Lab-1`, зокрема файлами `lab-1.md` та `README.md`, а також з наявними гілками. Для зручності копіювання структури та файлів, репозиторій курсу рекомендується склонувати на ваш ПК.

2. **Організація роботи в GitHub Projects**: Створіть картку для лабораторної роботи в проєкті курсу на GitHub за шаблоном `Lab-1-Ім'я-ПРІЗВИЩЕ`. Ім'я та прізвище мають бути англійською мовою, наприклад `Lab-1-Ivan-PULUJ`. Це правило діє для всіх подальших кроків.

3. **Створення та налаштування репозиторію**:

    - Створіть приватний репозиторій в організації `TNTU-F2-Software-Engineering` за шаблоном `Ім'я-ПРІЗВИЩЕ-SE201-2024-2025`.

- Перетворіть картку з попереднього кроку на `issue` та прив'яжіть до створеного репозиторію. Не змінюйте запропоновану назву!

4. **Налаштування локального середовища**:
   - Встановіть та налаштуйте Git, включно з SSH-ключем для роботи з GitHub.
   - Встановіть VS Code.
   - Створіть гілку для виконання лабораторної роботи безпосередньо з `issue` на GitHub.
   - Клонуйте репозиторій на локальний ПК та перейдіть у створену гілку.
   - **Зверніть увагу що усі подальші дії мають виконуватися в локальному репозиторії у гілці лабораторної роботи!**
5. **Структурування проєкту та аналіз**:
   - Створіть початкову структуру проєкту: каталоги `Lab-1` та Chat-Platform-MVP.
   - Додайте файл `README.md` в кореневий каталог, описавши в ньому призначення репозиторію.
   - Проведіть аналіз предметної області для MVP чат-платформи та запишіть ідеї у файл `Chat-Platform-MVP/docs/ideas.md`.
6. **Створення звіту та командна взаємодія**:
   - Створіть звіт у файлі `Lab-1/lab-1.md`, описуючи послідовність дій та використані команди Git.
   - Створіть `Pull Request` у гілку `main` та призначте одного зі студентів для рецензування (`Review`).
   - Після погодження виконайте `Squash and merge`, створіть тег (v0.1.0-lab1) та реліз. Додайте до релізу згенерований PDF-звіт.
7. **Генерація PDF-звіту**:
   - Згенеруйте `.pdf` версію звіту з markdown-файлу, використовуючи розширення VS Code, наприклад, *Markdown PDF*.

Додаткове завдання (дослідження)

Дослідіть тему кодування текстових файлів. У звіті в окремому розділі напишіть короткий огляд (3-4 абзаци), що розкриває наступні питання:

- Що таке кодування символів і чому воно важливе?
- Що таке UTF-8 і які його переваги?
- Які поширені проблеми виникають через неправильне кодування (наприклад, "кракозябри") і як їх уникати?

Для виконання завдання ознайомтесь з рекомендованими джерелами [1; 2].

Як перевірити свою роботу

Перш ніж здавати роботу, переконайтеся, що ви виконали всі вимоги. Використовуйте цей чек-ліст для самоперевірки:

- Репозиторій створено в організації `TNTU-F2-Software-Engineering` і має правильну назву.
- Створено `issue` з картки в GitHub Projects та прив'язано до вашого репозиторію.
- У репозиторії є гілка, створена з `issue`. Уся робота велася в цій гілці.
- Проєкт має правильну структуру: каталоги `Lab-1` та `Chat-Platform-MVP`.
- У корені проєкту є файл `README.md` з описом репозиторію.
- Файл `Chat-Platform-MVP/docs/ideas.md` містить ваші ідеї щодо MVP.
- Звіт `Lab-1/lab-1.md` створено, він детально описує ваші кроки та містить відповіді на теоретичні питання.
- Створено `Pull Request` для злиття гілки з роботою у `main`.
- Ваш PR пройшов рецензування (review) іншим студентом.
- Зміни було злито у `main` за допомогою `Squash and merge`.

- Створено тег (напр., `v0.1.0-lab1`) та реліз на GitHub.
- PDF-версія звіту згенерована та прикріплена як артефакт до релізу.

Як здати роботу

1. Переконайтеся, що всі пункти з розділу **«Як перевірити свою роботу»** виконано.
2. Завантажте артефакти релізу (PDF-звіт та посилання на репозиторій) у відповідну скриньку в системі дистанційного навчання.
3. Після успішної здачі переведіть картку в GitHub Projects у статус `Done` та закрийте відповідне `issue`.

Вітаємо, ви на крок ближче до того, щоб стати професійним розробником!

Рекомендовані джерела

Для глибшого розуміння матеріалу, розглянутого в цій лабораторній роботі, рекомендується ознайомитися з наступними ресурсами:

- Pro Git book (особливо розділи 1-3 про основи та гілкування) [3]
- Документація про базовий синтаксис Markdown на GitHub [4]
- Документація про GitHub Organizations [5]
- Документація про GitHub Repositories [6]
- Документація про GitHub Projects [7]
- Про Pull Requests [8]
- Коментування в Issue [9]
- The Absolute Minimum Every Software Developer Must Know About Unicode [1]
- Character encodings for beginners [2]

Список використаних джерел для лабораторної роботи №1

1. *Spolsky J.* The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!) — 2003. — https://www.joelonsoftware.com/2003/10/08/the-absolute-minimum-every-software-developer-absolutely-positively-must-know-about-unicode-and-character-sets-no-excuses/ (дата зверн. 14.09.2024).
2. *W3C.* Character encodings for beginners. — 2017. — https://www.w3.org/International/articles/serving-xhtml/character-encodings-for-beginners (дата зверн. 14.09.2024).
3. *Chacon S., Straub B.* Pro Git. — 2014. — https://git-scm.com/book/en/v2 (дата зверн. 27.08.2024).
4. *GitHub, Inc.* Basic writing and formatting syntax. — 2024. — https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax (дата зверн. 27.08.2024).
5. *GitHub, Inc.* About organizations. — 2024. — https://docs.github.com/en/organizations/collaborating-with-groups-in-organizations/about-organizations (дата зверн. 27.08.2024).
6. *GitHub, Inc.* About repositories. — 2024. — https://docs.github.com/en/repositories/creating-and-managing-repositories/about-repositories (дата зверн. 27.08.2024).
7. *GitHub, Inc.* About projects. — 2024. — https://docs.github.com/en/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects (дата зверн. 27.08.2024).
8. *GitHub, Inc.* About pull requests. — 2024. — https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests (дата зверн. 27.08.2024).

9. *GitHub, Inc.* Commenting on an issue. — 2024. — https://docs.github.com/en/issues/tracking-your-work-with-issues/commenting-on-an-issue (дата зверн. 14.09.2024).

Лабораторна робота №2: Аналіз вимог та проєктування архітектури чат-платформи

Цілі

Після завершення цієї лабораторної роботи, ви зможете:

- **Аналізувати та синтезувати**: Збирати, аналізувати та структурувати вимоги до програмного продукту, перетворюючи їх на чіткі специфікації.
- **Застосовувати знання на практиці**: Використовувати методи аналізу вимог та проєктування для створення архітектури системи, зокрема за допомогою UML.
- **Реалізовувати фази життєвого циклу ПЗ**: Виконувати ключові етапи аналізу вимог та високорівневого проєктування, що є фундаментом для подальшої розробки.
- **Дотримуватися стандартів та застосовувати інструменти**: Використовувати нотацію UML для візуального моделювання системи (діаграми варіантів використання та класів).
- **Оформлювати програмну документацію**: Створювати специфікацію вимог у форматі User Stories та базову проєктну документацію.
- **Працювати в команді**: Використовувати GitHub для обговорення та узгодження вимог і архітектурних рішень через Pull Requests.

Основна мета — навчитися перетворювати ідеї на структуровані вимоги та проєктувати архітектуру програмної системи.

Теоретичне підґрунтя

- **Вимоги до ПЗ (Software Requirements):** Це опис того, що система повинна робити. Вимоги поділяються на *функціональні* (конкретні функції, наприклад, ”відправка повідомлень”) та *нефункціональні* (якість, продуктивність, безпека, наприклад, ”система має відповідати за <1 секунди”).

- **User Stories:** Це неформальний опис функції ПЗ природною мовою з точки зору кінцевого користувача. Класичний шаблон: ”Як <роль>, я хочу <можливість>, щоб <отримати вигоду>”. Це гнучкий інструмент для визначення вимог в Agile-розробці.

- **UML (Unified Modeling Language):** Це стандартизована мова графічного моделювання для візуалізації, специфікації, конструювання та документування програмних систем.

- **Діаграма варіантів використання (Use Case Diagram):** Описує взаємодію між користувачами (акторами) та системою. Вона показує, *хто* і *що* може робити в системі, не деталізуючи, *як* це відбувається.

- **Діаграма класів (Class Diagram):** Описує статичну структуру системи: її класи, атрибути, методи та зв’язки між класами (асоціація, успадкування, композиція). Це ”креслення” вашої майбутньої програми.

- **Документація як код (Docs-as-Code):** Підхід, за якого документація створюється та керується тими ж інструментами, що й код (Git, Markdown, Pull Requests). Це забезпечує її актуальність та легку інтеграцію в процес розробки.

Що потрібно зробити

Виконайте завдання послідовно, дотримуючись командних процесів, описаних у лабораторній роботі №1.

1. **Організація роботи в GitHub Projects**: Створіть картку для лабораторної роботи за шаблоном `Lab-2-Ім’я-ПРІЗВИЩЕ` та перетворіть її на `issue` у вашому репозиторії. Уся подальша робота має вестися у гілці, створеній з цього `issue`.

2. **Аналіз вимог до чат-платформи**:

    - На основі ідей з `Chat-Platform-MVP/docs/ideas.md` визначте ключових користувачів (акторів) та їхні цілі.
    - Створіть файл Chat-Platform-MVP/docs/Chat-Platform-SRS.md (Software Requirements Specification).

- У цьому файлі опишіть щонайменше 5-7 ключових функцій у форматі **User Stories**.
- Визначте 3-4 важливі **нефункціональні вимоги** (наприклад, щодо продуктивності, надійності, зручності використання).

3. **Проєктування архітектури за допомогою UML**:
   - Створіть **Use Case діаграму**, що візуалізує взаємодію акторів із системою на основі ваших User Stories.
   - Створіть **Class діаграму**, що моделює основні сутності системи (наприклад, `User`, `Message`, `ChatRoom`) та зв'язки між ними.
   - Для створення діаграм використовуйте синтаксис **Mermaid** безпосередньо у вашому файлі `Chat-Platform-SRS.md`. Це дозволяє реалізувати підхід "документація як код" (docs-as-code), де діаграми є текстовими ресурсами, версіонуються разом із кодом і автоматично відображаються на платформах, таких як GitHub. Такий підхід спрощує підтримку документації в актуальному стані та усуває необхідність у зовнішніх редакторах та експорті зображень.
4. **Створення звіту та командна взаємодія**:
   - Створіть каталог `Lab-2` та файл звіту `Lab-2/lab-2.md`.
   - У звіті опишіть послідовність своїх дій, обгрунтуйте прийняті проєктні рішення та додайте створені User Stories і UML-діаграми.
   - Створіть `Pull Request` у гілку `main`, призначте рецензента та виконайте злиття після погодження (`Squash and merge`).
   - Створіть тег (`v0.2.0-lab2`) та реліз, додавши до нього згенерований PDF-звіт.

Додаткові завдання

1. Ознайомтеся з розділом 4.4. Model-Based Requirements Specification у `CHAPTER 01 Software Requirements` зі збірника знань SWEBOK Version 4[1]. Занотуйте ключові аспекти цього підходу.

2. Дослідіть поняття: *Software System*, *Software Platform*, *Middleware*, та *Application*. Проаналізуйте їхні взаємозв'язки та наведіть приклади для кожного [2].

Як перевірити свою роботу

Використовуйте цей чек-ліст для самоперевірки:

- Робота велася у гілці, створеній з `issue` для лабораторної роботи №2.
- Створено файл `Chat-Platform-MVP/docs/Chat-Platform-SRS.md`.
- У файлі специфікації є щонайменше 5 User Stories та 3 нефункціональні вимоги.
- Створено та збережено у відповідному каталозі Use Case та Class діаграми.
- Звіт `Lab-2/lab-2.md` містить опис роботи, обґрунтування рішень та всі необхідні артефакти (текст вимог, зображення діаграм).
- Створено Pull Request, він пройшов рецензування та був коректно злитий у `main`.
- Створено тег та реліз `v0.2.0-lab2` з прикріпленим PDF-звітом.

Як здати роботу

Процес здачі аналогічний до попередньої лабораторної роботи. Завантажте артефакти релізу (PDF-звіт та посилання на репозиторій) у відповідну скриньку в системі дистанційного навчання та переведіть картку в GitHub Projects у статус `Done`.

Рекомендовані джерела

- Розділи про аналіз вимог та проєктування з підручника Sommerville, I. "Software Engineering" [2].
- Cohn, M. "User Stories Applied: For Agile Software Development" [3].

- Fowler, M. “UML Distilled: A Brief Guide to the Standard Object Modeling Language” [4].
- Стаття про діаграми варіантів використання від Мартіна Фаулера [5].
- Документація GitHub про створення діаграм [6].
- Розділи про архітектуру ПЗ та системні моделі з підручника Sommerville, I. “Software Engineering” [2] для дослідження понять Software System, Platform, Middleware, Application.

Список використаних джерел для лабораторної роботи №2

1. *IEEE Computer Society*. Guide to the Software Engineering Body of Knowledge (SWEBOK V4.0). — Washington, D.C. : IEEE Computer Society, 2024. — ISBN 979-8-3503-0233-4.
2. *Sommerville I.* Software Engineering. — 10th. — Pearson, 2016. — https://software-engineering-book.com/.
3. *Cohn M.* User Stories Applied: For Agile Software Development. — Addison-Wesley Professional, 2004. — ISBN 978-0321205681.
4. *Fowler M.* UML Distilled: A Brief Guide to the Standard Object Modeling Language. — 3rd. — Addison-Wesley Professional, 2003. — ISBN 978-0321193681.
5. *Fowler M.* Use Case. — 2006. — https://martinfowler.com/bliki/UseCases.html (дата зверн. 14.09.2024).
6. *GitHub, Inc.* Creating diagrams. — 2024. — https://docs.github.com/en/get-started/writing-on-github/working-with-advanced-formatting/creating-diagrams (дата зверн. 14.09.2025).

Лабораторна робота №3: Конструювання та тестування ключового модуля

Цілі

Після завершення цієї лабораторної роботи, ви зможете:

- **Застосовувати знання на практиці**: Реалізувати ключовий функціонал програмної системи, перетворюючи проєктні рішення на робочий код.
- **Реалізовувати фази життєвого циклу ПЗ**: Виконати етапи конструювання та тестування, що є центральними у процесі розробки.
- **Застосовувати інструменти та принципи інженерії ПЗ**: Використовувати мову програмування, середовище розробки та фреймворк для тестування для створення якісного програмного продукту.
- **Забезпечувати якість коду**: Писати чистий, зрозумілий код та покривати його автоматизованими тестами для забезпечення надійності.
- **Працювати в команді**: Використовувати Git та GitHub для спільної розробки, рецензування коду (Code Review) та інтеграції змін.

Основна мета — отримати практичні навички написання, тестування та інтеграції коду в рамках командного проєкту.

Теоретичне підґрунтя

- **Конструювання ПЗ (Software Construction):** Це процес створення робочого програмного забезпечення через комбінацію кодування, верифікації, модульного тестування, інтеграційного тестування та зневадження.
- **Модульне тестування (Unit Testing):** Це вид тестування, що полягає в перевірці окремих компонентів (модулів, функцій, класів) програмного коду. Мета — переконатися, що кожна частина програми працює коректно ізольовано від інших.

- **Фреймворки для тестування (Testing Frameworks):** Інструменти, що надають структуру та набір функцій для написання й запуску тестів (наприклад, JUnit для Java, PyTest для Python, Jest для JavaScript).
- **Розробка через тестування (Test-Driven Development, TDD):** Методологія розробки, за якою спочатку пишуться тести, що описують бажану поведінку функціоналу, а потім — код, який проходить ці тести. Цикл: ”Червоний” (тест не проходить) -> ”Зелений” (тест проходить) -> ”Рефакторинг” (покращення коду).
- **Рецензування коду (Code Review):** Процес, під час якого інші розробники перевіряють вихідний код на наявність помилок, відповідність стандартам кодування та загальну якість перед його інтеграцією в основну кодову базу. Це ключова практика для підвищення якості ПЗ та обміну знаннями в команді.

Що потрібно зробити

1. **Організація роботи в GitHub Projects**: Створіть картку для лабораторної роботи за шаблоном `Lab-3-Ім'я-ПРІЗВИЩЕ` та перетворіть її на `issue`. Уся подальша робота має вестися у гілці, створеній з цього `issue`.
2. **Вибір технологічного стеку**: Оберіть мову програмування та технології, які ви будете використовувати для реалізації MVP чат-платформи (наприклад, Python + Flask/FastAPI, JavaScript + Node.js/Express, Java + Spring Boot). Обґрунтуйте свій вибір у звіті.
3. **Аналіз аналогічних рішень з відкритим кодом**: Оберіть один із запропонованих проєктів для аналізу: *Mattermost, Rocket.Chat, Zulip, Matrix (Element), Jitsi Meet*. Проаналізуйте його за такими критеріями та занотуйте результати у звіті:
    - **Документація та вимоги**: Наскільки чітко визначено цільову аудиторію, вимоги, сценарії використання (*use cases*), користувацькі історії (*user stories*) та дорожні карти (*roadmaps*).
    - **Архітектура**: Проаналізуйте архітектурні схеми, потоки даних, програмний стек, ключові протоколи та обґрунтування їх вибору.

- **Життєвий цикл та практики розробки**: Дослідіть структуру репозиторію, історію комітів, управління завданнями (*issues*), процеси *code review* (*pull requests*), тестування, розгортання та використання інструментів CI/CD.
- **Використання AI-інструментів**: Якщо ви залучали AI-асистентів для аналізу, опишіть, чому обрали конкретний інструмент, які можливості використовували та наведіть приклади запитів (промптів).

4. **Створення структури проєкту**: Перед початком кодування, створіть рекомендовану структуру каталогів у `Chat-Platform-MVP/`:

   - `src/`: для всього вихідного коду вашого додатку.
   - `tests/`: для автоматизованих тестів.
   - `docs/`: для всієї проєктної документації (вимоги, діаграми), яка вже частково створена.

   Така структура є стандартною практикою в індустрії, що робить проєкт зрозумілим та легким для навігації.

5. **Реалізація ключового функціоналу**:

   - Розмістіть вихідний код у каталозі `src/`.
   - На основі діаграми класів з попередньої лабораторної роботи створіть базові моделі даних (наприклад, класи `User`, `Message`).
   - Реалізуйте основний функціонал, визначений у User Stories (наприклад, реєстрація користувача, відправка та отримання повідомлень у чаті). На цьому етапі можна використовувати прості структури даних в пам'яті замість бази даних.

6. **Покриття коду модульними тестами**:

   - Розмістіть файли з тестами у каталозі `tests/`.
   - Оберіть та налаштуйте фреймворк для модульного тестування, що відповідає вашому технологічному стеку.
   - Напишіть unit-тести для реалізованого функціоналу. Переконайтеся, що тести перевіряють як успішні сценарії, так і обробку помилок.

- Налаштуйте запуск тестів у вашому проєкті.

7. **Створення звіту та командна взаємодія**:
   - Створіть каталог `Lab-3` та файл звіту `Lab-3/lab-3.md`.
   - У звіті опишіть обраний стек, процес реалізації, наведіть ключові фрагменти коду та написані тести. Обґрунтуйте, чому ваші тести є достатніми для перевірки функціоналу.
   - Створіть `Pull Request` у гілку `main`. Призначте рецензента для проведення **Code Review**. Виправте зауваження (якщо вони будуть) та виконайте злиття після погодження.
   - Створіть тег (`v0.3.0-lab3`) та реліз, додавши до нього згенерований PDF-звіт.

Додаткові завдання

1. Ознайомтеся з розділом CHAPTER 04 Software Construction та `CHAPTER 05 Software Testing` зі збірника знань SWEBOK Version 4[1].
2. Дослідіть принципи SOLID та спробуйте застосувати один або два з них у вашому коді. Коротко опишіть у звіті, як саме ви це зробили.
3. Дослідіть сучасні підходи до розробки програмного забезпечення з використанням штучного інтелекту (AI-Assisted Development). Проаналізуйте, як інструменти на кшталт GitHub Copilot змінюють процеси конструювання, тестування та зневадження. У звіті опишіть ключові переваги та потенційні ризики використання AI в повсякденній роботі програміста[2; 3].

Як перевірити свою роботу

- Робота велася у гілці, створеній з `issue` для лабораторної роботи №3.
- У репозиторії наявний вихідний код реалізованого функціоналу.
- Проєкт містить модульні тести, які можна запустити.

- Звіт `Lab-3/lab-3.md` містить опис роботи, вибір стеку, фрагменти коду та тестів.
- Створено Pull Request, він пройшов рецензування коду та був коректно злитий у `main`.
- Створено тег та реліз `v0.3.0-lab3` з прикріпленим PDF-звітом.

Як здати роботу

Завантажте артефакти релізу (PDF-звіт та посилання на репозиторій) у відповідну скриньку в системі дистанційного навчання та переведіть картку в GitHub Projects у статус `Done`.

Рекомендовані джерела

- Martin, R. C. “Clean Code: A Handbook of Agile Software Craftsmanship” [4].
- Beck, K. “Test Driven Development: By Example” [5].
- Документація до обраного вами фреймворку для тестування (PyTest, Jest, JUnit тощо).

Список використаних джерел для лабораторної роботи №3

1. *IEEE Computer Society*. Guide to the Software Engineering Body of Knowledge (SWEBOK V4.0). — Washington, D.C. : IEEE Computer Society, 2024. — ISBN 979-8-3503-0233-4.
2. *Bird S.*, *Wang C.*, *others sixteen*. Taking Flight with Copilot: A Study of Developer-AI Pair Programming // ACM Queue. — 2022. — Т. 20, № 5. — С. 48—77. — DOI: 10.1145/3571728.3571733.
3. A Longitudinal Study of How Developers Use AI-Based Code Generators / L.-W. Fakhoury [та ін.] // Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (ICSE 2024). — New York, NY, USA : ACM, 2024. — DOI: 10.1145/3597503.3639131.

4. *Martin R. C.* Clean Code: A Handbook of Agile Software Craftsmanship. — Prentice Hall, 2008. — ISBN 978-0132350884.

5. *Beck K.* Test Driven Development: By Example. — Addison-Wesley Professional, 2002. — ISBN 978-0321146533.

Лабораторна робота №4: Аналіз професійних стандартів та етики

Цілі

Після завершення цієї лабораторної роботи, ви зможете:

- **Застосовувати знання у практичних ситуаціях**: Проаналізувати вимоги ринку праці та створити професійний профіль, що відповідає сучасним стандартам.
- **Діяти на основі етичних міркувань**: Розуміти та застосовувати основні принципи кодексу професійної етики інженера-програміста.
- **Вчитися та оволодівати сучасними знаннями**: Визначити ключові навички, необхідні для кар'єрного зростання, та важливість навчання протягом усього життя.
- **Застосовувати професійні стандарти**: Ознайомитися зі стандартами IEEE та ISO, що регулюють галузь програмної інженерії.
- **Документувати та презентувати результати**: Створити якісний звіт та професійні профілі в соціальних мережах для демонстрації своїх досягнень.
- **Здійснювати пошук та аналіз інформації**: Дослідити ринок праці, визначити актуальні технології та вимоги до розробників.

Основна мета — підготувати вас до професійної діяльності, навчити аналізувати ринок праці та створити інструменти для успішного кар'єрного старту.

Теоретичне підґрунтя

- **Професійна практика в програмній інженерії (Software Engineering Professional Practice):** Це аспект програмної інженерії, що стосується знань, навичок та професіоналізму, які виходять за межі технічних аспектів. Він охоплює професіоналізм, групову динаміку, комунікативні навички та етичні норми (SWEBOK KA 14).

- **Професійні стандарти (Professional Standards):** Документи, що встановлюють інженерні та технічні вимоги до процесів, продуктів та послуг. В програмній інженерії ключовими є стандарти від **IEEE** (Institute of Electrical and Electronics Engineers) та **ISO** (International Organization for Standardization).

- **Кодекс професійної етики (Code of Ethics):** Набір принципів, розроблений для допомоги професіоналам у веденні бізнесу чесно та з доброчесністю. Найвідомішим є **Software Engineering Code of Ethics and Professional Practice**, розроблений ACM та IEEE-CS.

- **Професійний профіль розробника:** Це сукупність інформації про ваші навички, досвід, освіту та проєкти, представлена у форматах, привабливих для роботодавців (наприклад, резюме, профілі на GitHub та LinkedIn).

Що потрібно зробити

1. **Організація роботи в GitHub Projects**: Створіть картку для лабораторної роботи за шаблоном `Lab-4-Ім'я-ПРІЗВИЩЕ` та перетворіть її на `issue`. Уся подальша робота має вестися у гілці, створеній з цього `issue`.

2. **Аналіз професійних стандартів та етики**:

    - Ознайомтеся з **Software Engineering Code of Ethics and Professional Practice**.
    - У звіті виберіть три, на вашу думку, найважливіші принципи з кодексу та поясніть, чому ви вважаєте їх ключовими для сучасного розробника.
    - Наведіть приклад гіпотетичної ситуації, де порушення одного з цих принципів може призвести до негативних наслідків.

3. **Створення та актуалізація професійного профілю**:

    - **GitHub Profile**: Створіть або оновіть файл `README.md` у вашому профілі GitHub. Додайте інформацію про себе, ваші інтереси, технологічний стек та закріпіть 2-3 найважливіші проєкти.

- **LinkedIn Profile**: Створіть або оновіть свій профіль у LinkedIn. Заповніть розділи про освіту, досвід (можна вказати навчальні проєкти), навички та додайте коротке резюме (About).

4. **Аналіз ринку праці**:

   - Проаналізуйте 3-5 вакансій для Junior Software Engineer (або схожих) на українському або міжнародному ринку праці (напр., DOU, Djinni, LinkedIn Jobs).
   - У звіті визначте 5-7 ключових технічних навичок (мови, фреймворки) та 3-5
     ”м’яких” навичок (soft skills), які найчастіше вимагаються роботодавцями.
   - Порівняйте вимоги вакансій з вашими поточними навичками та складіть короткий план, що вам потрібно вивчити або вдосконалити в найближчий рік.

5. **Складання резюме для однієї обраної позиції**:

   - Підготуйте резюме з урахуванням вимог і специфіки позиції (якщо немає реального досвіду – можна вказати навчальні проєкти, курси, участь у хакатонах або волонтерську діяльність).
   - Результат – новий файл у каталозі `Lab-4` у форматі Markdown.
   - Зверніть увагу на структуру: контактні дані, освіта, досвід (проєкти), навички, додаткова інформація (сертифікати, досягнення).
   - За основу варто взяти свій актуалізований профіль у LinkedIn.
   - Поясніть, чому саме так назвали файл з резюме.

6. **Створення звіту та завершення роботи**:

   - Створіть каталог `Lab-4` та файл звіту `Lab-4/lab-4.md`.
   - У звіті задокументуйте результати аналізу стандартів, етики та ринку праці. Додайте посилання на ваші оновлені профілі GitHub та LinkedIn.

- Створіть `Pull Request` у гілку `main`. Оскільки ця робота є індивідуальною і не містить змін у коді проєкту, самостійно виконайте злиття після створення PR.
- Створіть тег (`v0.4.0-lab4`) та реліз, додавши до нього згенерований PDF-звіт.

Додаткові завдання

1. Ознайомтеся з розділом CHAPTER 14 Software Engineering Professional Practice зі збірника знань SWEBOK Version 4[1].

2. Дослідіть, що таке **impostor syndrome** (синдром самозванця) серед розробників. Напишіть у звіті кілька порад, як з ним боротися.

3. Проаналізуйте кар'єрний шлях відомого інженера-програміста (наприклад, Мартін Фаулер, Кент Бек, Лінус Торвальдс). Які ключові рішення та проєкти вплинули на їхню кар'єру?

4. **Аналіз етичної дилеми**. Уявіть (або візьміть готовий приклад) ситуацію, в якій компанія просить розробника свідомо «обійти» механізми безпеки для швидкого випуску продукту.

    - Оцініть ситуацію з точки зору Етичного кодексу. Що має робити програміст? Які можуть бути наслідки?
    - Сформулюйте коротку письмову відповідь (до 1 сторінки), у якій викладіть свою позицію та пропозицію вирішення з урахуванням етичних норм.
    - *Для глибшого аналізу можна звернутися до реальних прикладів, як-от скандал "Dieselgate" компанії Volkswagen, де програмне забезпечення використовувалося для обману тестів на викиди, або випадок з апаратом для променевої терапії Therac-25, де програмні помилки призвели до смерті пацієнтів. Рекомендована література для роздумів на цю тему — книга Роберта Мартіна "The Clean Coder" [2].*

Як перевірити свою роботу

- Робота велася у гілці, створеній з `issue` для лабораторної роботи №4.
- Звіт `Lab-4/lab-4.md` містить аналіз етичних принципів, ринку праці та посилання на профілі.
- Профілі на GitHub та LinkedIn оновлені та виглядають професійно.
- Створено Pull Request, який був коректно злитий у `main`.
- Створено тег та реліз `v0.4.0-lab4` з прикріпленим PDF-звітом.

Як здати роботу

Завантажте артефакти релізу (PDF-звіт та посилання на репозиторій) у відповідну скриньку в системі дистанційного навчання та переведіть картку в GitHub Projects у статус `Done`.

Рекомендовані джерела

- Software Engineering Code of Ethics and Professional Practice [3].
- Посібник з оформлення профілю GitHub [4].
- Поради щодо створення профілю LinkedIn для студентів [5].

Список використаних джерел для лабораторної роботи №4

1. *IEEE Computer Society*. Guide to the Software Engineering Body of Knowledge (SWEBOK V4.0). — Washington, D.C. : IEEE Computer Society, 2024. — ISBN 979-8-3503-0233-4.
2. *Martin R. C.* The Clean Coder: A Code of Conduct for Professional Programmers. — Prentice Hall, 2011. — ISBN 978-0137081073.

3. *ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices*. Software Engineering Code of Ethics and Professional Practice. — 1999. — https://ethics.acm.org/code-of-ethics/software-engineering-code/ (дата зверн. 14.09.2024).

4. *GitHub, Inc.* Managing your profile README. — 2024. — https://docs.github.com/en/account-and-profile/setting-up-and-managing-your-github-profile/managing-your-profile-readme (дата зверн. 14.09.2024).

5. *Arruda W.* How To Create The Perfect LinkedIn Profile For College Students. — 2018. — https://www.forbes.com/sites/williamarruda/2018/10/09/how-to-create-the-perfect-linkedin-profile-for-college-students/ (дата зверн. 14.09.2024).

Додаток А. Приклад оформлення звіту лабораторної роботи

Титульний аркуш звіту

Звіт про виконання лабораторної роботи повинен містити титульний аркуш наступного формату:

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Тернопільський національний технічний університет
імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем
та програмної інженерії

Кафедра програмної інженерії

ЗВІТ

про виконання лабораторнї роботи № __

з дисципліни

«Основи програмної інженерії (SE201)»

Тема: _____

Виконав(ла): _____
_____
____

Перевірив(ла): _____
____

Тернопіль — 2025

Рекомендована структура звіту

Звіт повинен містити такі основні розділи:

1. Вступ

- Мета виконання лабораторної роботи
- Актуальність проблеми, що розглядається
- Задачі роботи (перелік із нумерацією)

2. Теоретичні основи

- Короткий виклад теоретичних концепцій програмної інженерії, необхідних для розв'язання задачі
- Основні визначення та принципи
- Огляд використаних методів та підходів програмної інженерії

3. Практична реалізація

- Описання алгоритму розв'язання
- Вихідний код (або посилання на Jupyter notebook в репозиторії)
- Технічні особливості реалізації
- Використані бібліотеки та інструменти
- Архітектурні рішення (за потреби)

4. Результати та аналіз

- Результати експериментів або тестування
- Візуалізація (графіки, таблиці, діаграми)
- Інтерпретація результатів
- Порівняння з очікуваними результатами

- Аналіз якості програмного продукту (за потреби)

5. Висновки

- Підсумування основних результатів лабораторної роботи
- Відповідь на поставлені в роботі питання та задачі
- Рекомендації для подальшого розвитку або удосконалення
- Аналіз труднощів, з якими довелося мати справу під час виконання
- Оцінка ефективності застосованих методів програмної інженерії

6. Список посилань

- Список всіх використаних джерел літератури
- Рекомендується використовувати стиль цитування IEEE
- Включити посилання на використані бібліотеки, фреймворки та інструменти
- Посилання на стандарти та методології програмної інженерії

Контрольний список для звіту

Перед здачею переконайтеся, що звіт містить:

- [ ] Титулка з усіма необхідними реквізитами
- [ ] Вступ з ясною постановкою задачі
- [ ] Теоретичні основи методів, що використовуються
- [ ] Опис алгоритму та реалізації
- [ ] Результати експериментів з візуалізацією
- [ ] Аналіз та інтерпретація результатів
- [ ] Висновки, що узагальнюють проведену роботу

– Список посилань на використані джерела

– Послідовність сторінок відповідно до змісту

– Правильне оформлення таблиць, рисунків та рівнянь

– Послідовність нумерації та посилань

– Відсутність граматичних та синтаксичних помилок

– Посилання на вихідний код в репозиторії

Додаток Б. Матриця відповідності SWEBOK Knowledge Areas

SWEBOK KA	Лб1	Лб2	Лб3	Лб4
1. Software Requirements		+		
2. Software Architecture		+		
3. Software Design		+		
4. Software Construction	+		+	
5. Software Testing			+	
6. Software Engineering Operations	+			
7. Software Maintenance				
8. Software Configuration Management	+			
9. Software Engineering Management	+			
10. Software Engineering Process				
11. Software Engineering Models & Methods		+		
12. Software Quality			+	
13. Software Security				
14. Software Engineering Professional Practice				+
15. Software Engineering Economics				
16. Computing Foundations	+			
17. Mathematical Foundations				
18. Engineering Foundations				

Табл. 1: Матриця відповідності лабораторних робіт до областей знань SWEBOK