МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ



МЕТОДИ ЗАБЕЗПЕЧЕННЯ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Методичні вказівки до лабораторних робіт
для здобувачів третього (освітньо-наукового) рівня вищої освіти
освітньо-наукової програми «Інженерія програмного забезпечення»
галузі знань F Інформаційні технології
спеціальності F2 Інженерія програмного забезпечення

Луцьк 2026

УДК 005.8
М 41

Голова вченої ради факультету комп’ютерних та інформаційних технологій _____ І. С. Кондіус

Електронна копія друкованого видання передана для внесення в репозиторій ЛНТУ
Директор бібліотеки _____ Н. П. Поліщук

Затверджено вченою радою факультету комп’ютерних та інформаційних технологій ЛНТУ, протокол № ____ від «____» _____ 2026 року.

Розглянуто і схвалено на засіданні кафедри інженерії програмного забезпечення ЛНТУ, протокол № ____ від «____» _____ 2026 року.

Завідувач кафедри ІПЗ _____ Н. М. Ліщина

Укладач: _____ В. О. Ліщина, к.т.н., доцент, завідувач кафедри комп’ютерних наук ЛНТУ

Рецензент: _____ О. О. Гордєєв, д.т.н., професор, професор кафедри інженерії програмного забезпечення ЛНТУ

М 41

Методи забезпечення якості програмного забезпечення: методичні вказівки до лабораторних робіт для здобувачів третього (освітньо-наукового) рівня вищої освіти освітньо-наукової програми «Інженерія програмного забезпечення» галузі знань F Інформаційні технології спеціальності F2 Інженерія програмного забезпечення / уклад.: В. О. Ліщина. Луцьк: ЛНТУ, 2026. 28 с.

Методичні вказівки містять вісім лабораторних робіт із побудови моделей якості, вимірювання та емпіричного оцінювання якості програмного забезпечення, статичного аналізу, тестування й автоматизованих перевірок. До кожної роботи подано мету, завдання, теоретичні відомості, порядок виконання, приклади та контрольні запитання.

В. О. Ліщина, 2026

ЗМІСТ

Загальні вказівки 4
Лабораторна робота 1 5
Лабораторна робота 2 7
Лабораторна робота 3 10
Лабораторна робота 4 12
Лабораторна робота 5 15
Лабораторна робота 6 17
Лабораторна робота 7 20
Лабораторна робота 8 22
Рекомендовані джерела 26

Загальні вказівки

Методичні вказівки призначені для виконання восьми лабораторних робіт з ОК8 «Методи забезпечення якості програмного забезпечення». Загальний обсяг лабораторних занять – 30 академічних годин. Працюйте з невеликим програмним компонентом або навчальним прикладом і пояснюйте отримані результати з урахуванням умов перевірки.

Організація роботи

Виконуйте роботи індивідуально. За погодженням із викладачем використовуйте об’єкт власного дисертаційного дослідження, якщо обсяг завдань залишається посильним для заняття. Для робіт 3–6 можна повторно використовувати навчальний компонент Python. У роботі 8 використайте матеріали однієї з попередніх робіт.

Подання результатів – електронне: таблиця, вихідні файли, автоматично сформовані журнали перевірок або посилання на навчальний репозиторій. Завершуйте роботу коротким усним захистом. Окремий друкований звіт не потрібний. Якщо завдання передбачає стислий звіт, оформіть його електронно у README.md або в документі з таблицею результатів. Висновки записуйте безпосередньо під таблицею результатів або в README.md – достатньо 3–5 речень.

Зберігайте файли в папках lab01–lab08. У кожній папці вкажіть своє прізвище, об’єкт, дату виконання та використані версії інструментів. Для відтворення програмних робіт збережіть команду запуску та перелік залежностей. Додавайте посилання на запозичені матеріали, зазначайте власний внесок і використання засобів ШІ.

Тематичний план

№	Тема лабораторної роботи	Години	Основний електронний результат
1	Побудова контекстної моделі якості програмної системи.	4	Таблиця моделі якості
2	Розроблення та перевірка плану вимірювання якості програмного забезпечення.	4	План і пробні вимірювання
3	Емпіричне порівняння методів або інструментів забезпечення якості програмного забезпечення.	4	Протокол і таблиця порівняння
4	Оцінювання якості коду та технічного боргу засобами статичного аналізу.	4	Код до і після зміни, результати аналізу
5	Порівняння тестування на основі прикладів і властивостей із використанням мутаційного аналізу.	4	Два набори тестів і таблиця мутантів
6	Налаштування автоматизованих перевірок якості та безпеки у конвеєрі CI/CD.	4	Конфігурація та результати запусків

№	Тема лабораторної роботи	Години	Основний електронний результат
7	Оцінювання окремих характеристик якості інтелектуального компонента або розподіленої програмної системи.	2	Дані базового та зміненого сценаріїв
8	Удосконалення та емпірична перевірка методу забезпечення якості програмного забезпечення.	4	Порівняння початкової та зміненої процедури

Підготовка середовища

Для робіт 1–2 використовуйте табличний редактор. Для робіт 3–8 підготуйте Python 3.12, редактор коду та термінал. Для роботи 6 потрібен навчальний проєкт GitLab із доступним Runner. Отримайте доступ до нього до початку заняття.

Створіть робочу папку і віртуальне середовище. У Windows виконайте наведені команди без активації середовища. Надалі запускайте інтерпретатор цього середовища або оберіть його в редакторі коду.

```
py -3.12 -m venv .venv
.venv\Scripts\python -m pip install pytest hypothesis pylint radon
.venv\Scripts\python -m pip freeze > requirements-lock.txt
.venv\Scripts\python --version
```

У Linux або macOS використовуйте python3.12 замість py -3.12 та .venv/bin/python замість .venv\Scripts\python. У подальших командах python означає інтерпретатор обраного віртуального середовища. Зафіксуйте встановлені версії; під час порівняння варіантів їх не змінюйте.

Перевірка й оцінювання

Під час захисту відкрийте результати, відтворіть одну перевірку та поясніть вибір метрик, висновок і обмеження. Оцінювання здійснюється за робочою програмою дисципліни. Враховуються повнота виконання, коректність методу й обчислень, відтворюваність, самостійність і обґрунтованість висновків. Відсутність очікуваного покращення є допустимим результатом коректно проведеного експерименту.

У навчальних прикладах пороги прийнятності задаються умовою завдання. Для власного об’єкта обґрунтуйте їх окремо. Не змінюйте пороги після отримання результатів без пояснення причин і повторної перевірки.

Лабораторна робота 1

Тема. Побудова контекстної моделі якості програмної системи

Тривалість – 4 академічні години.

Мета. Сформувати здатність обґрунтовано визначати характеристики якості програмної системи відповідно до потреб зацікавлених сторін і контексту використання.

Засоби – табличний або текстовий редактор, браузер. Результат – одна електронна таблиця з описом контексту, моделлю якості та коротким висновком.

Завдання

1. Обрати програмну систему або компонент, визначити його призначення, зацікавлені сторони та основний сценарій використання.
2. Обрати 3–5 пріоритетних характеристик якості за ISO/IEC 25010:2023 та ISO/IEC 25019:2023, пояснивши їхню доречність.
3. Сформулювати вимоги до обраних характеристик, визначити метрики та критерії прийнятності.
4. Установити, які дані або результати перевірок потрібні для підтвердження виконання вимог.
5. Оформити таблицю «потреба – характеристика якості – вимога – метрика – критерій – доказ» та визначити обмеження побудованої моделі.

Теоретичні відомості

Контекст використання описує користувачів, їхні завдання, середовище й обмеження. Від нього залежить вибір характеристик якості та способів перевірки. ISO/IEC 25010:2023 задає модель якості продукту, а ISO/IEC 25019:2023 — модель якості під час використання. Зіставляйте властивості продукту з результатами його застосування для зацікавлених сторін [1, 2].

Вимога визначає очікувану поведінку, метрика – спосіб її вимірювання, критерій прийнятності – допустиме значення, доказ – матеріал, на якому ґрунтується висновок. Наприклад, «швидкий пошук» потрібно уточнити через операцію, обсяг даних, умови та час відповіді.

Навчальний об’єкт

Побудуйте модель для одного сценарію обраної системи. Визначте 3–5 пріоритетних характеристик, сформулюйте перевірювані вимоги та заплануйте отримання доказів. Реалізація програмної системи для цієї роботи не потрібна.

Порядок виконання

1. Оберіть об’єкт: сервіс запису на консультацію, бібліотечний каталог, систему звернень, застосунок обліку завдань або компонент власного дослідження. Виділіть один основний сценарій.
2. Опишіть призначення, 2–3 зацікавлені сторони, пристрій, умови мережі та очікувану інтенсивність використання. Зазначте функції, які не входять до оцінювання.
3. Оберіть 3–5 характеристик із моделей [1, 2]. Для кожної зазначте стандарт і поясніть зв’язок із потребою конкретної сторони. Використовуйте назви характеристик або підхарактеристик відповідної редакції стандарту.
4. Сформулюйте для кожної обраної характеристики одну вимогу. Вкажіть метрику, одиницю вимірювання, умови перевірки та критерій прийнятності.
5. Визначте потрібний доказ: протокол функціональних перевірок, журнал часу відповіді, результати виконання користувацького завдання тощо. Позначте його статус: «заплановано» або «отримано».
6. Перевірте кожний зв’язок: потреба → характеристика → вимога → метрика → критерій → доказ. Уточніть формулювання, які допускають різне тлумачення.

7. Змініть одну умову контексту, наприклад кількість користувачів або якість зв’язку. Поясніть, яку вимогу чи перевірку потрібно переглянути.

Навчальний приклад

Об’єкт – сервіс запису на консультацію. Користувач обирає вільний час і отримує підтвердження. Оцінювання охоплює бронювання та доступ до нього; платежі й інтеграції з іншими сервісами не розглядаються.

Потреба	Характеристика	Вимога та умови	Метрика і критерій	Доказ
Уникнути подвійного запису	Функціональна придатність, ISO/IEC 25010	Відхиляти повторний запис на зайнятий час	0 подвійних бронювань у 10 перевірках	Таблиця очікуваних і фактичних результатів
Швидко отримати відповідь	Ефективність виконання, ISO/IEC 25010	Підтвердження за одного активного користувача	Максимум часу серед 10 повторень ≤ 2 с	Журнал вимірювань і опис середовища
Захистити власний запис	Захищеність, ISO/IEC 25010	Заборонити зміну чужого бронювання	0 успішних змін у 5 перевірках доступу	Перевірки з двома навчальними обліковими записами

Числові критерії в таблиці є умовами цього навчального прикладу. Додайте до пояснення моделі наслідки для користувача: чи може він завершити запис у зазначених умовах та якими будуть наслідки помилки. Для власної моделі перевірте ці питання за ISO/IEC 25019:2023.

Подання результатів і захист

Збережіть quality_model.xlsx або таблицю в іншому погодженому форматі. На першому аркуші опишіть контекст, на другому подайте 3–5 рядків моделі. Під таблицею зазначте обмеження й результат зміни контексту. Під час захисту поясніть один повний ланцюжок від потреби до доказу.

Контрольні запитання

1. Чим відрізняються метрика та критерій прийнятності?
2. Чому одна модель якості не підходить для всіх умов використання?
3. Які висновки дозволяє зробити запланована перевірка і чого вона не підтверджує?
4. Чим якість продукту відрізняється від якості під час використання?

Джерела: [1–3].

Лабораторна робота 2

Тема. Розроблення та перевірка плану вимірювання якості програмного забезпечення

Тривалість – 4 академічні години.

Мета. Сформувати навички добору метрик якості, організації вимірювань та обґрунтованої інтерпретації отриманих результатів.

Засоби – табличний редактор; за бажанням Python. Результат – електронна таблиця з планом, початковими даними й обчисленнями.

Завдання

1. Для обраної програмної системи або компонента сформулювати одну ціль вимірювання та 2–3 запитання за підходом GQM.
2. Визначити 3–5 метрик, указати одиниці вимірювання, джерела даних і порядок їх збирання.
3. Установити критерії прийнятності та правила інтерпретації значень метрик. За потреби передбачити нормування показників.
4. Виконати пробне вимірювання на невеликому наборі даних або програмному прикладі та перевірити можливість повторного отримання результатів.
5. Проаналізувати можливі похибки, дублювання показників і ризики хибної інтерпретації; за результатами перевірки уточнити план вимірювання.

Теоретичні відомості

Підхід Goal–Question–Metric (GQM) пов’язує ціль оцінювання із запитаннями й метриками [4]. Наприклад, ціль «оцінити стабільність відповіді сервісу» конкретизується запитаннями про час відповіді та частку помилок. Кожна метрика повинна мати джерело даних, одиницю, правило обчислення та умови застосування.

Середнє значення показує загальний рівень показника, медіана менш чутлива до окремих великих значень, максимум характеризує найгірше спостереження у вибірці. Частка помилок дорівнює кількості невдалих запитів, поділеній на загальну кількість запитів. Зазначайте, чи враховуєте час невдалих запитів у показниках затримки.

Навчальний об’єкт

Сформулюйте одну ціль, 2–3 запитання та 3–5 метрик. Виконайте пробне вимірювання або перевірте обчислення на наведених навчальних даних. Для власного об’єкта використайте модель роботи 1.

Початкові дані

Скопіюйте таблицю в редактор. Це синтетичний журнал для перевірки методики обчислення. Час виміряно від надсилання запиту до отримання відповіді або завершення очікування; усі 10 рядків належать одному сценарію.

Запит	Час, мс	Успіх, 1 або 0
1	120	1
2	135	1
3	110	1
4	160	1
5	125	1
6	140	1
7	500	0
8	130	1
9	150	1
10	115	1

Порядок виконання

1. Запишіть ціль: «Оцінити час і успішність виконання запитів у навчальному сценарії». Сформулюйте запитання про типовий час, найгірше спостереження та частку невдалих запитів.
2. Оберіть чотири метрики: медіану часу, максимальний час, частку помилок і частку своєчасних успішних відповідей. Своєчасною вважайте успішну відповідь за час не більше 200 мс.
3. До обчислення результатів запишіть критерії: медіана ≤ 150 мс; максимум ≤ 500 мс; частка помилок ≤ 5%; частка своєчасних успішних відповідей ≥ 95%. Для власного сценарію погодьте інші обґрунтовані критерії.
4. У плані зазначте формулу, одиницю, джерело й порядок збирання даних. Для часових метрик у цьому прикладі використовуйте всі запити, включно з невдалим.
5. Виконайте обчислення. Частки подавайте у відсотках: кількість відповідних рядків / 10 × 100%. Визначте, які критерії виконано.
6. Повторіть обчислення іншим способом: формулою редактора, вручну або коротким скриптом. Порівняйте значення, перевірте одиниці й правила округлення.
7. Додатково обчисліть середній час лише успішних запитів. Поясніть, чому цей показник не замінює частку помилок. Не вилучайте невдалий запит із початкового журналу.
8. Уточніть план: додайте позначення статусу, правило врахування тайм-аутів і кількість спостережень. Запишіть одне обмеження вибірки з 10 запитів.

Шаблон плану вимірювання

Запитання	Метрика й одиниця	Формула або правило	Дані та умови	Критерій
Який типовий час?	Медіана, мс	Медіана всіх значень часу	10 запитів одного сценарію	≤ 150 мс
Яка частка невдач?	Частка помилок, %	Невдалі / усі × 100%	Статус кожного запиту	≤ 5%

Доповніть таблицю двома іншими метриками. Не об’єднуйте різні характеристики в один бал без обґрунтування ваг.

Самоперевірка

Для наведеного журналу медіана часу всіх запитів становить 132,5 мс, максимум – 500 мс, частка помилок – 10%, частка своєчасних успішних відповідей – 90%. Якщо значення відрізняються, перевірте діапазони клітинок і знаменники часток.

Подання результатів і захист

Подайте measurements.xlsx з аркушами «План», «Дані» та «Результати». Збережіть формули в клітинках. Додайте 3–5 речень про виконання критеріїв, похибки та уточнення плану. На захисті покажіть обчислення однієї метрики.

Контрольні запитання

1. Як метрика пов’язана з ціллю в підході GQM?
2. Як тайм-аут впливає на інтерпретацію часу відповіді?

3. Чому середній час успішних запитів не показує загальну надійність?
4. Чим повторне обчислення відрізняється від нового вимірювання?

Джерела: [3, 4, 5].

Лабораторна робота 3

Тема. Емпіричне порівняння методів або інструментів забезпечення якості програмного забезпечення

Тривалість – 4 академічні години.

Мета. Сформувати навички планування та проведення відтворюваного порівняльного експерименту для оцінювання засобів забезпечення якості.

Засоби – Python, Pylint, редактор коду. Результат – файл прикладу, дві конфігурації запуску, журнали й таблиця порівняння.

Завдання

1. Сформулювати дослідницьке запитання та обрати два методи, інструменти або конфігурації одного інструмента для порівняння.
2. Визначити спільний набір програмних прикладів або даних, показники порівняння та умови проведення експерименту.
3. Підготувати короткий протокол експерименту із зазначенням версій інструментів, налаштувань і послідовності дій.
4. Провести вимірювання; для показників, що залежать від випадковості або умов виконання, передбачити повторні запуски. Подати результати в таблиці або на графіку.
5. Порівняти результати, оцінити практичне значення виявлених відмінностей та описати обмеження експерименту. Зберегти дані й налаштування, необхідні для його повторення.

Теоретичні відомості

Порівнюйте варіанти на одному об’єкті й за однакових умов. Базовий варіант – початковий підхід, з яким зіставляють альтернативу. Висновок повинен відповідати обраній метриці: кількість повідомлень аналізатора не дорівнює кількості підтверджених дефектів [5, 6].

Для цієї роботи порівняйте дві конфігурації Pylint: A – лише категорія помилок; B – стандартний набір перевірок. Розрізняйте помилки виконання, попередження про ризики й рекомендації стилю. Підтверджуйте зміст кожного вибраного повідомлення переглядом коду.

Початковий приклад

Створіть файл sample.py. За вимогою функція average має повертати середнє непорожнього списку чисел; порожній список повинен спричиняти ValueError. Функція describe має повертати текстове подання аргументу. Код містить навмисні недоліки.

```
import math
def average(values):
    total = sum(values)
    count = len(values)
    unused = 10
    return total / count
def describe(value):
    return str(unknown_value)
```

Порядок виконання

1. Сформулюйте запитання: «Як розширення набору правил впливає на кількість і зміст корисних повідомлень?» Очікування запишіть до запуску.
2. Складіть власний перелік проблем коду. Обов'язково перевірте порожній список і виконання describe. Відокремте порушення функціональних вимог від недоліків оформлення.
3. Запишіть у protocol.txt версію Python і Pylint, назву файла, команди та дату. Збережіть незмінну копію sample.py.
4. Запустіть конфігурації A і B. Збережіть результати у два текстові файли. Ненульовий код завершення Pylint за наявності повідомлень є очікуваним.

```
python -m pylint --version
python -m pylint --errors-only sample.py > analysis_A.txt
python -m pylint --reports=no --score=no sample.py > analysis_B.txt
```

5. Для кожного варіанта порахуйте всі повідомлення й окремо повідомлення про підтверджені проблеми. Наведіть щонайменше три приклади з кодом правила, рядком і поясненням.
6. Перевірте, чи виявив аналізатор порушення вимоги для порожнього списку. Виконайте контрольні виклики нижче; зафіксуйте фактичні винятки. Помилку виконання не вважайте правильною обробкою входу, якщо її тип не відповідає вимозі.

```
python -c "from sample import average; print(average([]))"
python -c "from sample import describe; print(describe(5))"
```

7. Повторіть статичний аналіз тією самою командою. За незмінного коду очікується той самий набір повідомлень. Вимірювання часу та статистичні тести в базовому завданні не потрібні.
8. Порівняйте користь і обсяг ручного перегляду. Запишіть, яку конфігурацію обрали б для швидкої перевірки, а яку – для докладного аналізу, та чому.

Таблиці результатів

Показник	A, лише помилки	B, стандартні правила
Усього повідомлень	Заповніть	Заповніть
Повідомлення про підтверджені проблеми	Заповніть	Заповніть
Виявлені порушення функціональних вимог	Заповніть	Заповніть
Недоліки, які потрібно перевірити тестами	Заповніть	Заповніть
Правило і рядок	Суть повідомлення	Підтвердження або обмеження
Заповніть	Заповніть	Покажіть відповідний фрагмент коду

Повідомлення про невикористаний імпорт може бути правильним, але не свідчить про збій програми. Невиявлена логічна помилка показує межі обраного аналізу. Висновки поширюйте лише на перевірений приклад і конфігурації.

Подання результатів і захист

Подайте sample.py, protocol.txt, analysis_A.txt, analysis_B.txt та таблицю. У висновку поясніть одну перевагу й одне обмеження кожного варіанта. На захисті повторіть аналіз і прокоментуйте одне повідомлення.

Контрольні запитання

1. Чому потрібно використовувати однаковий програмний приклад?
2. Чи означає більша кількість повідомлень кращий аналіз?
3. Як відрізнити хибне спрацьовування від правильної, але малопріоритетної рекомендації?
4. Які властивості цього прикладу потребують динамічної перевірки?

Джерела: [5, 6].

Лабораторна робота 4

Тема. Оцінювання якості коду та технічного боргу засобами статичного аналізу

Тривалість – 4 академічні години.

Мета. Набути навичок застосування статичного аналізу та критичного оцінювання його результатів для визначення проблем якості коду й пріоритетів їх усунення.

Засоби – Python, Pylint, Radon, pytest. Результат – версії коду до і після зміни, тести та таблиця проблем.

Завдання

1. Обрати невеликий програмний проєкт або наданий викладачем репозиторій, зафіксувати його версію та налаштувати засіб статичного аналізу.
2. Отримати показники складності, дублювання коду, порушень правил або інструментальної оцінки технічного боргу.
3. Детально розглянути 3–5 виявлених проблем, пояснити їхній можливий вплив на якість програмного забезпечення та перевірити наявність хибних спрацьовувань.
4. Визначити пріоритети усунення проблем і виправити одну з них без зміни очікуваної поведінки програми.
5. Повторити аналіз і виконати наявні тести. Порівняти результати та пояснити, які висновки про якість коду можна зробити на їхній основі.

Теоретичні відомості

Статичний аналіз досліджує код без виконання основного сценарію програми. Цикломатична складність характеризує структуру розгалужень; вона не є кількістю дефектів. Технічний борг пов’язують із рішеннями, які ускладнюють подальші зміни та потребують майбутніх витрат. Пріоритет виправлення визначайте за впливом, імовірністю проблеми та трудомісткістю [6, 7].

Рефакторинг змінює внутрішню структуру зі збереженням очікуваної поведінки. Виправлення функціонального дефекту оцінюйте окремо: воно навмисно змінює неправильну поведінку.

Початковий компонент

Створіть quality.py. Функція clamp обмежує ціле число інтервалом [low, high]. Вона приймає лише значення типу int, відхиляє bool та інші типи через TypeError, a low > high — через ValueError.

```
import math
def clamp(x, low=0, high=100):
    if any(type(v) is not int for v in (x, low, high)):
        raise TypeError("integers required")
    if low > high:
        raise ValueError("invalid interval")
    if x < low:
        return low
    else:
        if x > high:
            return high
        else:
            return x
```

Порядок виконання

1. Збережіть початковий файл як before/quality.py. Робочу копію залиште в корені lab04. Створіть test_quality.py за зразком нижче й доповніть перевірками обох меж, low = high та некоректного типу.

```
import pytest
from quality import clamp
def test_inside():
    assert clamp(30, 0, 100) == 30
def test_outside():
    assert clamp(-1, 0, 100) == 0
    assert clamp(101, 0, 100) == 100
def test_invalid_interval():
    with pytest.raises(ValueError):
        clamp(5, 10, 0)
```

2. Запустіть тести, Pylint і Radon. Збережіть журнали. Якщо команда radon недоступна, перевірте вибір віртуального середовища: виконуваний файл розміщений у його каталозі Scripts або bin.

```
python -m pytest -q test_quality.py
python -m pylint --reports=no quality.py > before_pylint.txt
radon cc quality.py -s -a > before_complexity.txt
```

3. Розгляньте 3–5 повідомлень або недоліків: невикористаний імпорт, відсутність опису призначення, зайву вкладеність тощо. Для кожного вкажіть вплив, пріоритет і орієнтовні витрати у хвилинах. Не подавайте власну оцінку часу як розрахунок інструмента.

4. Оберіть одну структурну проблему. Наприклад, замініть зайві else після return послідовними перевірками. Не змінюйте правила обробки типів та інтервалу.

5. Повторно запустіть ті самі тести й аналізатори. Збережіть журнали з префіксом after_. Якщо тести впали, з’ясуйте причину й виправте зміну.

6. Порівняйте кількість відповідних повідомлень, цикломатичну складність і результати тестів. Поясніть користь зміни навіть тоді, коли числове значення складності залишилося тим самим.

7. Укажіть одну проблему, яку залишили без виправлення, та поясніть її пріоритет. Зазначте, які властивості не покривають наявні тести.

Таблиця проблем

Проблема або правило	**Вплив на якість**	**Пріоритет**	**Оцінка витрат, хв**	**Рішення**
Заповніть 3–5 рядків	Поясніть для цього коду	Високий, середній або низький	Власна оцінка	Виправлено або відкладено

Подання результатів і захист

Подайте before/quality.py, оновлений quality.py, test_quality.py, журнали й таблицю. Додайте коротке пояснення зміни. На захисті покажіть різницю версій і повторний запуск тестів.

Контрольні запитання

1. Чому зменшення кількості попереджень не гарантує правильності програми?
2. Чим рефакторинг відрізняється від виправлення дефекту?
3. Як визначити пріоритет усунення проблеми?
4. Чому після спрощення вкладеності складність може не змінитися?

Джерела: [6–8].

Лабораторна робота 5

Тема. Порівняння тестування на основі прикладів і властивостей із використанням мутаційного аналізу

Тривалість – 4 академічні години.

Мета. Сформувати навички розроблення тестів на основі прикладів і властивостей та оцінювання їхньої ефективності за допомогою мутаційного тестування.

Засоби – Python, pytest, Hypothesis. Результат – два набори тестів, п’ять мутантів і таблиця результатів.

Завдання

1. Обрати невелику функцію або програмний компонент із чітко визначеною очікуваною поведінкою.
2. Розробити набір тестів на основі конкретних прикладів, включаючи типові, граничні та некоректні вхідні дані.
3. Сформулювати 2–3 властивості об’єкта тестування та реалізувати їх перевірку із застосуванням автоматичної генерації вхідних даних.
4. Виконати мутаційний аналіз обох наборів тестів на спільному наборі мутантів. Зафіксувати частку виявлених мутантів, час виконання та, за наявності відповідного інструмента, покриття коду.
5. Проаналізувати кілька мутантів, які не були виявлені тестами, з урахуванням можливої еквівалентності початковій програмі. Порівняти можливості й обмеження двох підходів та запропонувати покращення тестів.

Теоретичні відомості

Тест на основі прикладу задає конкретний вхід і очікуваний результат. Тест на основі властивості перевіряє загальне твердження на автоматично згенерованих входах. Мутант – версія програми з однією контрольованою зміною. Мутант виявлений, якщо початкова програма проходить набір тестів, а змінена версія спричиняє змістовну невдачу тесту [8–10].

Мутаційний показник $MS = K / (M - E) \times 100\%$, де K – кількість виявлених мутантів, M – кількість коректно виконаних мутантів, E – кількість обґрунтовано еквівалентних. Збої середовища та синтаксично некоректні зміни обліковуйте окремо. Для навчальної вибірки з п’яти заданих нееквівалентних мутантів знаменник дорівнює 5.

Об’єкт і тести

Використайте clamp із роботи 4 після рефакторингу. Збережіть початкову справну копію original_quality.py. Усі порівняння виконуйте на тих самих п’яти мутантах. Застосовуйте по одній зміні до справної версії.

Створіть test_examples.py з 6–8 конкретними перевірками: значення всередині інтервалу, нижче й вище меж, рівність межам, вироджений інтервал і некоректні входи. Для винятків використовуйте pytest.raises. Кожна перевірка має відповідати специфікації роботи 4.

Створіть test_properties.py. Наведено дві властивості. Додайте третю: для x від 0 до 100 результат clamp(x, 0, 100) дорівнює x.

```
from hypothesis import given, settings, strategies as st
from quality import clamp
@settings(max_examples=50, derandomize=True)
@given(st.integers(min_value=-1000, max_value=1000))
def test_range(x):
    assert 0 <= clamp(x, 0, 100) <= 100
@settings(max_examples=50, derandomize=True)
@given(st.integers(min_value=-1000, max_value=1000))
def test_idempotence(x):
    y = clamp(x, 0, 100)
    assert clamp(y, 0, 100) == y
```

Мутації

Мутант	Одна зміна у справному коді	Контрольний вхід для ручної перевірки
M1	У гілці x < low замініть return low на return low + 1	x = −1, low = 0, high = 100
M2	У гілці x > high замініть return high на return high − 1	x = 101, low = 0, high = 100
M3	Останній return x замініть на return x + 1	x = 50, low = 0, high = 100
M4	Перевірку type(v) is not int замініть на not isinstance(v, int)	x = True, low = 0, high = 100
M5	raise ValueError у перевірці low > high замініть на return low	x = 5, low = 10, high = 0

Порядок виконання

1. Запустіть обидва набори на початковій програмі. Спершу усуньте всі помилки в тестах і коді. Зафіксуйте версії пакетів та параметри Hypothesis.

```
python -m pytest -q test_examples.py
python -m pytest -q test_properties.py
```

2. Створіть M1 у робочому quality.py. Запустіть два набори окремими командами. Збережіть журнал кожного запуску з назвою мутанта та набору.
3. Відновіть справний quality.py і лише після цього створіть M2. Аналогічно перевірте M3–M5. Зберігайте копії мутантів у папці mutants; не запускайте їх усі як один компонент.
4. Для кожного запуску зафіксуйте статус «виявлено» або «не виявлено» та тривалість із підсумкового рядка pytest. Якщо тест не запустився через помилку імпорту чи середовища, виправте причину й повторіть запуск.
5. Порахуйте MS для кожного набору. Для часу візьміть медіану трьох запусків кожного набору на справній програмі. Не порівнюйте час невдалого раннього завершення з часом повної перевірки як однакові обсяги роботи.
6. Розгляньте щонайменше два невиявлені мутанти, якщо вони є. Перевірте контрольні входи з таблиці. Визначте, якої властивості або прикладу бракує. Для M1–M5 наведені входи демонструють відмінність від початкової програми.
7. Додайте одну перевірку, яка усуває конкретний пропуск, і повторіть аналіз відповідного мутанта. Збережіть початкові результати окремо від покращених.
8. Відновіть справний quality.py. Зробіть висновок про переваги, обмеження та взаємне доповнення двох наборів.

Таблиця результатів

Мутант	Приклади	Властивості	Причина пропуску або назва тесту
M1–M5, окремий рядок	Виявлено / ні	Виявлено / ні	Поясніть
Підсумок	K і MS	K і MS	Обсяг вибірки — 5

Якщо властивості використовують тільки цілі x і фіксований правильний інтервал, вони не перевіряють відхилення bool або low > high. Урахуйте цю межу генерації у висновку.

Подання результатів і захист

Подайте справний компонент, два набори тестів, копії п’яти мутантів, журнали та таблицю. На захисті покажіть один мутант, який виявляється тестом, і поясніть одну властивість. Автоматизацію через mutmut можна використати додатково за погодженням із викладачем.

Контрольні запитання

1. Чому початкова програма має проходити всі тести?
2. Чи завжди невиявлений мутант є еквівалентним?
3. Як область генерації впливає на результат property-based тестування?
4. Чому один мутаційний показник недостатній для загального висновку про якість тестів?

Джерела: [5, 8–10].

Лабораторна робота 6

Тема. Налаштування автоматизованих перевірок якості та безпеки у конвеєрі CI/CD

Тривалість – 4 академічні години.

Мета. Набути навичок інтеграції перевірок якості й безпеки у процес збирання програмного забезпечення та визначення умов прийняття програмних змін.

Засоби – навчальний GitLab із Runner, Python, pytest, Pylint. Результат – репозиторій із конфігурацією та три завершені запуски конвеєра.

Завдання

1. Використати підготовлений репозиторій із базовою конфігурацією CI/CD та визначити послідовність автоматизованих перевірок.
2. Налаштувати запуск тестів і статичного аналізу коду, а також одну перевірку безпеки: аналіз залежностей, пошук секретів або перевірку вебзастосунку.
3. Установити умови проходження перевірок і блокування змін, визначити порядок збереження звітів. Для перевірок вебзастосунку обрати доречні вимоги OWASP ASVS.
4. Внести контрольовану помилку або використати наданий навчальний приклад із відомою проблемою та перевірити реакцію конвеєра.
5. Зіставити реалізовані перевірки з однією або двома практиками OWASP SAMM, визначити наявні докази їх виконання та запропонувати наступний крок удосконалення процесу.

Теоретичні відомості

Конвеєр послідовно виконує перевірки після зміни коду. Quality gate визначає, які результати дозволяють прийняти зміну. Машинний журнал або JUnit XML є доказом виконання перевірки. Для запобігання злиттю змін налаштуйте в навчальному проєкті вимогу успішного pipeline перед merge [11].

У роботі поєднайте тести, статичний аналіз та пошук заданого навчального маркера секрету. Вузьке правило пошуку демонструє принцип блокування небезпечної зміни; його область дії – файли Python у папці src. Для зіставлення процесу використайте практику Security Testing моделі OWASP SAMM [12].

Підготовка репозиторію

Створіть структуру src/quality.py, tests/test_quality.py, tools/check_secrets.py, requirements-lock.txt і .gitlab-ci.yml. У src/quality.py розмістіть справний clamp. До src додайте порожній __init__.py. У тестах замініть імпорт на from src.quality import clamp. Візьміть конкретні тести роботи 5 без навмисних мутантів.

Створіть requirements-lock.txt у чистому середовищі Python 3.12 після встановлення pytest та pylint. У файл мають входити ці пакети та їх залежності з точними версіями. Додайте файл до репозиторію. Перед запуском переконайтеся, що Runner може завантажити образ Python і пакети.

Перевірка навчального маркера

Створіть tools/check_secrets.py. Скрипт перевіряє наявність маркера тільки в src/*.py та вкладених папках. Використовуйте виключно штучний рядок LAB_SECRET_123; реальні паролі й токени не потрібні.

```
from pathlib import Path
import sys
marker = "LAB_SECRET_" + "123"
found = []
for path in Path("src").rglob("*.py"):
    if marker in path.read_text(encoding="utf-8"):
        found.append(str(path))
Path("reports").mkdir(exist_ok=True)
message = "PASS" if not found else "FOUND: " + ", ".join(found)
Path("reports/security.txt").write_text(message, encoding="utf-8")
print(message)
sys.exit(1 if found else 0)
```

Конфігурація конвеєра

Створіть .gitlab-ci.yml із наведеним вмістом. Відступи задавайте пробілами.

```
image: python:3.12-slim
stages: [test, analysis, security]
before_script:
  - python -m pip install -r requirements-lock.txt
  - mkdir -p reports
tests:
  stage: test
  script:
    - python -m pytest -q tests --junitxml=reports/tests.xml
  artifacts:
    when: always
    paths: [reports/]
    reports:
      junit: reports/tests.xml
static_analysis:
  stage: analysis
  script:
    - python -m pylint --errors-only src > reports/pylint.txt
  artifacts:
    when: always
    paths: [reports/]
security_check:
  stage: security
  script:
    - python tools/check_secrets.py
  artifacts:
    when: always
    paths: [reports/]
```

Порядок виконання

1. Виконайте всі три команди перевірок локально. Створіть reports перед запуском pytest. Переконайтеся, що тести проходять, Pylint не знаходить помилок категорії E/F, а пошук маркера завершується PASS.
2. Додайте файли до навчального репозиторію. Перевірте YAML у редакторі конвеєра GitLab. Запустіть pipeline для справної версії; збережіть посилання та ідентифікатор commit.
3. Зафіксуйте правила прийняття: усі тести проходять; статичний аналіз у вибраному режимі має код завершення 0; навчальний маркер у src відсутній. Увімкніть вимогу успішного pipeline перед злиттям у налаштуваннях merge requests; за відсутності прав зверніться до викладача.
4. Створіть окрему гілку й додайте до src/quality.py коментар # LAB_SECRET_123. Це не змінює поведінку функції. Запустіть конвеєр повторно та перевірте невдачу security_check.
5. Відкрийте артефакт reports/security.txt. Переконайтеся, що вказано файл зі штучним маркером. Зіставте фактичну реакцію з правилом прийняття.
6. Видаліть навчальний маркер, створіть новий commit і отримайте успішний pipeline. Запишіть три ідентифікатори запусків: початковий, із проблемою, після усунення.
7. Зіставте налаштування з практикою OWASP SAMM Security Testing: вкажіть автоматизовану перевірку, момент виконання, доказ і подальшу дію. Запропонуйте один наступний крок, наприклад перевірку залежностей спеціалізованим засобом.

Фіксація результатів

Версія	Тести	Статичний аналіз	Перевірка маркера	Pipeline / commit
Початкова	Заповніть	Заповніть	Заповніть	Посилання
Із маркером	Заповніть	Заповніть	Заповніть	Посилання
Після усунення	Заповніть	Заповніть	Заповніть	Посилання

Якщо завдання має статус pending, перевірте доступність Runner. Якщо невдача сталася під час завантаження залежностей, це збій підготовки середовища, а не виявлення проблеми коду. Збережіть і проаналізуйте відповідний журнал.

Подання результатів і захист

Надайте посилання на репозиторій, три pipeline та коротку таблицю результатів у README.md. На захисті відкрийте невдалу перевірку й поясніть умову блокування. Скріншоти потрібні лише тоді, коли викладач не має доступу до журналів.

Контрольні запитання

1. Чим відрізняються невдала перевірка коду й збій середовища?
2. Яка умова реально забороняє merge, якщо pipeline не пройдено?
3. Чому пошук одного маркера не є повною перевіркою секретів?
4. Які докази виконання практики SAMM отримано?

Джерела: [6, 8, 11, 12].

Лабораторна робота 7

Тема. Оцінювання окремих характеристик якості інтелектуального компонента або розподіленої програмної системи

Тривалість – 2 академічні години.

Мета. Сформувати навички перевірки специфічних характеристик якості програмних систем в умовах зміни вхідних даних, навантаження або доступності компонентів.

Засоби – Python і табличний редактор. Результат – CSV двох сценаріїв та коротка таблиця порівняння.

Завдання

1. Обрати один із запропонованих викладачем готових прикладів: компонент машинного навчання або розподілену програмну систему.
2. Визначити одну або дві характеристики якості, відповідні метрики та критерії прийнятності.
3. Підготувати контрольований сценарій: для компонента машинного навчання – зміну розподілу або спотворення вхідних даних; для розподіленої системи – зміну навантаження чи імітацію відмови компонента.
4. Виконати перевірку за базових і змінених умов, зібрати та порівняти значення обраних показників.

5. Пояснити виявлені зміни, визначити обмеження перевірки та сформулювати рекомендації щодо підвищення якості досліджуваного об’єкта.

Теоретичні відомості

Надійність і час відповіді сервісу залежать від доступності його компонентів. Показник рівня обслуговування (SLI) вимірює конкретну властивість, а ціль (SLO) задає потрібне значення за визначений період або набір запитів. У роботі оцініть частку успішних запитів і частку успішних відповідей у межах заданого часу [13].

Використовуйте готову локальну модель звернення до сервісу з залежним компонентом. Вона формує синтетичні значення часу й помилок; її результати характеризують заданий сценарій моделювання. Альтернативний готовий ML-компонент можна використати за погодженням із викладачем, обравши одну зміну входів і 1–2 метрики.

Готовий сценарій

Створіть scenario.py. У базовому режимі всі запити успішні. У зміненому кожне п’яте звернення завершується відмовою, а до часу решти додається 40 мс. Зафіксований тайм-аут – 500 мс. Код записує журнал у results.csv.

```
import csv
rows = []
for mode in ("base", "changed"):
    for request in range(1, 101):
        failed = mode == "changed" and request % 5 == 0
        latency = 80 + (request % 7) * 10
        if mode == "changed":
            latency += 40
        rows.append({
            "mode": mode,
            "request": request,
            "latency_ms": 500 if failed else latency,
            "success": int(not failed),
        })
with open("results.csv", "w", newline="", encoding="utf-8") as file:
    writer = csv.DictWriter(file, fieldnames=rows[0].keys())
    writer.writeheader()
    writer.writerows(rows)
```

Порядок виконання

1. Опишіть об’єкт як модель сервісу із залежним компонентом. Визначте дві характеристики: надійність виконання запитів та ефективність виконання.
2. Запишіть критерії: частка успішних запитів ≥ 95%; частка успішних відповідей не довше 150 мс ≥ 90%. Знаменник обох часток – усі 100 запитів відповідного сценарію.
3. Запустіть python scenario.py. Відкрийте results.csv у табличному редакторі й перевірте наявність 100 рядків для кожного mode.
4. Для base і changed окремо порахуйте кількість успіхів та кількість рядків, де success = 1 і latency_ms ≤ 150. Обчисліть частки й зіставте з критеріями.
5. Побудуйте просту стовпчикову діаграму двох часток для обох сценаріїв. Порівняйте, чи однаково змінилися надійність і своєчасність відповідей.
6. Змініть тільки частоту відмов: request % 5 == 0 замініть на request % 10 == 0. До повторного запуску збережіть початковий CSV під іншою назвою. Перевірте, як змінився результат.

7. Запропонуйте одну дію для підвищення якості: обмеження часу очікування, повторний запит або резервний компонент. Вкажіть можливу ціну цієї дії: додаткові запити, затримка чи витрати ресурсів.

Таблиця порівняння

Сценарій	Успішні / усі	Успіх, %	Своєчасні успішні / усі	Своєчасність, %	Виконання SLO
Базовий	Заповніть	Заповніть	Заповніть	Заповніть	Так / ні
Відмова кожного 5-го запиту	Заповніть	Заповніть	Заповніть	Заповніть	Так / ні
Відмова кожного 10-го запиту	Заповніть	Заповніть	Заповніть	Заповніть	Так / ні

Для самоперевірки: у базовому сценарії успішні всі 100 запитів; у початковому зміненому – 80. Частку своєчасних успішних відповідей обчисліть самостійно. Повторний запуск без зміни коду дає ті самі дані, оскільки модель детермінована.

Подання результатів і захист

Подайте scenario.py, початковий і змінений CSV та таблицю з діаграмою. У 3–5 реченнях поясніть причину змін, рекомендацію та обмеження моделі. На захисті поясніть знаменник однієї частки.

Контрольні запитання

1. Чим частка успішних запитів відрізняється від частки своєчасних успішних відповідей?
2. Чому відмови не можна вилучати зі знаменника SLI?
3. Яких властивостей реальної мережі не відтворює ця модель?
4. Який компроміс виникає під час повторних запитів?

Джерела: [1, 3, 13].

Лабораторна робота 8

Тема. Удосконалення та емпірична перевірка методу забезпечення якості програмного забезпечення

Тривалість – 4 академічні години.

Мета. Сформувати здатність обґрунтовувати локальне вдосконалення методу забезпечення якості та перевіряти його результативність у відтворюваному експерименті.

Засоби – матеріали попередніх робіт, Python або табличний редактор. Результат – два варіанти процедури, спільні дані та таблиця порівняння.

Завдання

1. Обрати окремий метод або процедуру забезпечення якості та визначити конкретну проблему, яку передбачається усунути.
2. Запропонувати одне обґрунтоване вдосконалення, наприклад зміну правила аналізу, добору тестів, порогового значення або способу інтерпретації метрики.

3. Описати початковий і вдосконалений варіанти, визначити спільні дані, показники порівняння та критерії оцінювання результату.
4. Провести порівняльний експеримент за однакових умов і подати отримані результати в таблиці або на графіку.
5. Сформулювати висновок про наявність або відсутність очікуваного покращення, зазначити обмеження та загрози валідності. Підготувати стислий звіт і матеріали для повторення експерименту.

Теоретичні відомості

Вдосконалення повинно усувати визначену проблему: пропуск граничних випадків, надлишок малокорисних повідомлень, нечітке правило прийняття тощо. Порівнюйте початковий варіант A та змінений B за однаковими показниками. Змінюйте один елемент, щоб пояснити його внесок [3, 5].

Відокремлюйте дані, на яких обрали зміну, від даних для її підсумкової перевірки. Результат на кількох навчальних прикладах обґрунтовує локальний висновок. Наукову новизну оцінюють додатково за результатами аналізу літератури.

Варіанти завдання

Варіант	**Початкова проблема**	**Одне можливе вдосконалення**	**Показники**
A, рекомендований	Тести пропускають помилки на межах	Додати правило обов’язкової перевірки меж і некоректних входів	Виявлені мутанти; кількість тестів; час
Б	Статичний аналіз формує надлишок малопріоритетних повідомлень	Уточнити набір правил або пріоритет перегляду	Підтверджені проблеми; обсяг ручного перегляду
В	Рішення про якість спирається тільки на середній час	Додати частку своєчасних успішних відповідей	Виявлені проблемні сценарії; помилкові рішення

Оберіть один варіант. Для Б або В заздалегідь підготуйте незалежний перелік відомих проблем чи критеріїв оцінювання, щоб не визначати «правильність» лише за власним новим правилом.

Порядок виконання

1. Запишіть проблему, об’єкт і дослідницьке запитання. Для варіанта A: «Чи підвищує явне включення меж і некоректних входів здатність тестів виявляти зміни поведінки clamp?»
2. Зафіксуйте варіант A з роботи 5. Опишіть правило B: додавати для кожного інтервалу перевірки low − 1, low, high, high + 1 та некоректних аргументів. Обмежте зміну 2–4 додатковими тестами або параметризованими групами.
3. Визначте критерій до підсумкового запуску: більша кількість виявлених нееквівалентних мутантів за збереження проходження справної програми. Окремо зафіксуйте приріст кількості перевірок і часу, прийнятний для вашого сценарію.

4. Для добору зміни використайте M1–M5 із роботи 5. Після фіксації B перевірте обидва варіанти на нових мутантах H1–H3 з таблиці нижче. Не змінюйте B за результатами H1–H3 до завершення первинного порівняння.
5. Спершу запустіть A та B на справному компоненті. Потім по черзі створюйте кожний новий мутант із початкової програми, запускайте обидва набори та зберігайте результати.
6. Для часу виконайте по три запуски A та B на справній програмі, чергуючи A–B–A–B–A–B. Використайте медіану. Версії середовища й обладнання залишайте незмінними.
7. Обчисліть різницю кількості виявлених мутантів і різницю мутаційних показників у відсоткових пунктах. Якщо A виявляє 1 із 3, а B – 2 із 3, різниця становить 33,3 відсоткового пункта.
8. Поясніть результат: покращення, відсутність відмінностей або погіршення. Опишіть витрати та щонайменше дві загрози валідності: малий приклад, обмежені мутації, залежність від генератора входів або вплив середовища.

Нові контрольовані зміни для варіанта A

Мутант	Зміна у справному clamp	Контрольна перевірка
H1	Замініть if low > high на if low >= high	clamp(5, 5, 5) має повернути 5
H2	Умову x < low замініть на x < low − 1	clamp(−1, 0, 100) має повернути 0
H3	Умову x > high замініть на x > high + 1	clamp(101, 0, 100) має повернути 100

Результати M1–M5 і H1–H3 подавайте окремо. Якщо всі нові мутанти виявлено вже варіантом A, зафіксуйте відсутність приросту на цьому наборі; порівняйте витрати та запропонуйте наступний експеримент.

Таблиця результатів

Показник	Варіант A	Варіант B	Різниця та інтерпретація
Виявлено H1–H3	Заповніть	Заповніть	Заповніть
Мутаційний показник, %	Заповніть	Заповніть	Відсоткові пункти
Кількість зібраних pytest тестів	Заповніть	Заповніть	Обсяг перевірки
Медіана часу трьох запусків, с	Заповніть	Заповніть	Витрати часу

Кількість тестів pytest і кількість згенерованих Hypothesis входів є різними величинами. Вкажіть налаштування генерації окремо; не підмінюйте ними кількість зібраних тестових функцій або параметризованих випадків.

Подання результатів і захист

Подайте файли A і B, контрольні дані або мутанти, журнали та таблицю. Стислий електронний звіт оформіть у README.md: зазначте проблему, зміну, команду відтворення, результати, висновок і обмеження – орієнтовно 8–10

речень та таблиця порівняння. На захисті поясніть, чому отриману відмінність можна або не можна пов’язати з вашим удосконаленням.

Контрольні запитання

1. Чому для порівняння потрібно зафіксувати базовий варіант?
2. Навіщо відокремлювати дані добору зміни від контрольних даних?
3. Як оцінити практичну користь покращення разом із його витратами?
4. Чому відсутність позитивного ефекту також є результатом дослідження?

Джерела: [3, 5, 8–10].

Рекомендовані джерела

1. ISO/IEC 25010:2023. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Product quality model. URL: https://www.iso.org/standard/78176.html.
2. ISO/IEC 25019:2023. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Quality-in-use model. URL: https://www.iso.org/standard/78177.html.
3. ISO/IEC 25040:2024. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Quality evaluation framework. URL: https://www.iso.org/standard/83467.html.
4. Dalton J. Goal, Question, Metric (GQM). Great Big Agile. Berkeley, CA: Apress, 2019. P. 177–179. DOI: https://doi.org/10.1007/978-1-4842-4206-3_33.
5. Felderer M., Travassos G. H. (eds.). Contemporary Empirical Methods in Software Engineering. Cham: Springer, 2020. X, 525 p. DOI: https://doi.org/10.1007/978-3-030-32489-6.
6. Pylint. Running Pylint. Офіційна документація. URL: https://pylint.readthedocs.io/en/stable/user_guide/usage/run.html.
7. Radon. Command-line Usage. Офіційна документація. URL: https://radon.readthedocs.io/en/latest/commandline.html.
8. pytest. How to invoke pytest. Офіційна документація. URL: https://docs.pytest.org/en/stable/how-to/usage.html.
9. Hypothesis. Quickstart. Офіційна документація. URL: https://hypothesis.readthedocs.io/en/latest/quickstart.html.
10. Papadakis M., Kintis M., Zhang J., Jia Y., Le Traon Y., Harman M. Mutation Testing Advances: An Analysis and Survey. Advances in Computers. 2019. Vol. 112. P. 275–378. DOI: https://doi.org/10.1016/bs.adcom.2018.03.015.
11. GitLab. CI/CD YAML syntax reference. Офіційна документація. URL: https://docs.gitlab.com/ci/yaml/.
12. OWASP SAMM. Security Testing. URL: https://owaspsamm.org/model/verification/security-testing/.
13. Beyer B., Murphy N. R., Rensin D. K., Kawahara K., Thorne S. (eds.). The Site Reliability Workbook: Practical Ways to Implement SRE. Sebastopol: O'Reilly Media, 2018. URL: https://sre.google/workbook/table-of-contents/.

Навчально-методичне видання

МЕТОДИ ЗАБЕЗПЕЧЕННЯ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Методичні вказівки до лабораторних робіт
для здобувачів третього (освітньо-наукового) рівня вищої освіти
освітньо-наукової програми «Інженерія програмного забезпечення»
галузі знань F Інформаційні технології
спеціальності F2 Інженерія програмного забезпечення

Укладач *В. О. Ліщина*

Комп'ютерний набір *В. О. Ліщина*

Редактор *В. О. Ліщина*

Підписано до друку «____» _____ 2026 р.
Формат А4. Папір офсетний. Гарнітура Times New Roman.
Ум. друк. арк. _____. Обл.-вид. арк. _____.
Тираж _____ прим. Замовлення № _____.

Відділ іміджу та промоції
Луцького національного технічного університету
43018, м. Луцьк, вул. Львівська, 75
Друк – ВІП ЛНТУ