МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ



МЕТОДИ ЗАБЕЗПЕЧЕННЯ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Конспект лекцій
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: конспект лекцій 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.

Конспект лекцій охоплює вісім тем із моделей і метрик якості програмного забезпечення, емпіричного оцінювання, статичного аналізу, тестування, DevSecOps та оцінювання інтелектуальних і розподілених систем. До кожної теми подано план, пояснення, навчальний приклад, зв'язок із лабораторною роботою та питання для самоперевірки.

В. О. Ліщина, 2026

ЗМІСТ

Вступ 4
Тема 1. Якість програмного забезпечення як об’єкт наукового дослідження 4
Тема 2. Метрики та методи вимірювання якості програмного забезпечення 6
Тема 3. Емпіричне оцінювання методів забезпечення якості 8
Тема 4. Статичний аналіз, архітектурна якість і технічний борг 11
Тема 5. Сучасні методи тестування та оцінювання ефективності тестів 13
Тема 6. Безперервне забезпечення якості, DevSecOps та експлуатаційний контроль 15
Тема 7. Особливості оцінювання якості інтелектуальних і розподілених систем 17
Тема 8. Розроблення та емпірична перевірка вдосконалення методу забезпечення якості 19
Основні терміни 23
Рекомендовані джерела 24

Вступ

Конспект лекцій з ОК08 «Методи забезпечення якості програмного забезпечення» призначений для здобувачів третього (освітньо-наукового) рівня вищої освіти за освітньо-науковою програмою «Інженерія програмного забезпечення». Матеріал розкриває вісім тем дисципліни та поєднує пояснення основних понять із прикладами дослідницьких рішень.

Забезпечення якості починається з визначення потреб і завершується обґрунтованим висновком про те, наскільки програмна система задовольняє ці потреби в заданих умовах. Окремий тест, показник або висновок аналізатора дає лише частину потрібних доказів. Завдання дослідника – встановити, що саме вимірюється, яким способом отримано результат і на які об’єкти та умови поширюється висновок.

Після опрацювання матеріалу здобувач має вміти пов’язувати вимоги до якості з показниками та доказами, планувати порівняння методів, інтерпретувати результати аналізу й тестування, а також обґрунтовувати локальне вдосконалення процедури забезпечення якості. Математичні пояснення спираються на відсотки, середні значення, медіану та прості порівняння. Статистичні поняття введено на рівні, потрібному для коректного планування дослідження.

Наскрізні навчальні приклади – сервіс бронювання консультацій і невеликий програмний компонент обмеження числового значення заданим інтервалом. Числові дані прикладів є умовними. Вони пояснюють спосіб міркування та не становлять результатів дослідження реального продукту.

У кожній темі наведено план, пояснення, навчальний приклад, зв’язок із лабораторною роботою та питання для самоперевірки. Посилання в квадратних дужках відповідають списку джерел цього конспекту. Використані редакції стандартів і документації зазначено в бібліографії.

Тема 1. Якість програмного забезпечення як об’єкт наукового дослідження

План: поняття якості; моделі якості; контекст використання; вимоги, метрики та докази; дослідницький підхід до оцінювання.

1.1 Поняття якості та його розвиток

Якість програмного забезпечення характеризує його здатність задовольняти визначені потреби зацікавлених сторін за встановлених умов. Функціонально правильний продукт може бути незручним, повільним або вразливим. Тому оцінювання якості охоплює як правильність результату, так і поведінку системи, можливість її супроводження та наслідки використання.

Початкове зосередження на пошуку помилок поступово доповнилося аналізом процесів розроблення, архітектури, користувацького досвіду та експлуатаційних даних. У сучасній роботі ці підходи співіснують. Тестування виявляє порушення поведінки, рецензування допомагає уточнити рішення до запуску програми, моніторинг показує проблеми реального використання.

Слід розрізняти забезпечення якості й контроль якості. Забезпечення якості охоплює організацію діяльності, яка створює обґрунтовану впевненість у виконанні вимог: планування, правила, відповідальність, перевірки й удосконалення процесу. Контроль якості безпосередньо встановлює відповідність конкретного результату вимогам. Тестування є одним із засобів такого контролю, але не замінює всієї роботи із забезпечення якості.

1.2 Моделі якості продукту та якості під час використання

Модель якості впорядковує характеристики, за якими описують і досліджують систему. ISO/IEC 25010:2023 визначає модель якості продукту; ISO/IEC 25019:2023 – модель якості під час використання; ISO/IEC 25002:2024 пояснює огляд і застосування моделей якості [1–3]. Модель не задає універсального прохідного бала для будь-якої системи. Характеристики потрібно конкретизувати відповідно до об’єкта оцінювання.

Для якості продукту розглядають, наприклад, функціональну придатність, ефективність виконання, надійність, захищеність і супроводжуваність. Функціональна придатність стосується того, чи забезпечує система потрібні функції. Ефективність виконання пов’язана з часом і ресурсами. Супроводжуваність важлива тоді, коли систему потрібно аналізувати, виправляти та змінювати.

Якість під час використання розглядають через взаємодію конкретних людей із системою в певному середовищі. Наприклад, технічно правильна форма бронювання не гарантує, що користувач зможе успішно записатися з мобільного пристрою за нестійкого зв’язку. Для оцінювання потрібен опис користувача, завдання та умов. Перехід до іншого контексту вимагає перегляду обґрунтування висновків [2].

Результат вимірювання окремої характеристики не слід називати загальною оцінкою якості. Якщо перевірено лише швидкість відповіді, ще немає доказів правильності бронювання або обмеження доступу до чужих записів. Повнота висновку визначається повнотою обраних вимог і перевірок.

1.3 Контекст і зацікавлені сторони

Контекст використання включає призначення системи, категорії користувачів, основні сценарії, навантаження, пристрої, мережеві умови та зовнішні обмеження. Зацікавленими сторонами можуть бути користувач, замовник, розробник, адміністратор і дослідник. Їхні пріоритети не завжди збігаються: користувач очікує швидкого результату, адміністратор – контрольованості, розробник – можливості безпечно змінювати код.

Для сервісу бронювання можна визначити такий контекст: один користувач обирає вільний час, підтверджує запис і отримує повідомлення; одночасно працюють до 50 користувачів. Платежі та зовнішні медичні системи не входять до меж цього прикладу. Без такого обмеження незрозуміло, які перевірки необхідні та чи є набір доказів достатнім.

1.4 Від потреби до доказу

Потребу необхідно перетворити на перевірювану вимогу. Формулювання «система має працювати швидко» не містить одиниці вимірювання,

навантаження та порога. Уточнена вимога може звучати так: «За 50 одночасних користувачів не менше 95 % запитів бронювання мають успішно завершуватися за час до 2 секунд». Значення порога в цьому прикладі є навчальним.

Потреба	**Перевірювана вимога**	**Показник і доказ**
Уникнути подвійного запису	Одночасні запити на один час не створюють два бронювання	Кількість подвійних записів; журнал конкурентного тесту
Отримати результат без тривалого очікування	Не менше 95 % запитів успішні та тривають до 2 с за заданого навантаження	Частка своєчасних успішних запитів; результати вимірювання
Обмежити доступ до чужих записів	Користувач не читає і не змінює запис іншого користувача	Результати позитивних і негативних перевірок авторизації

Метрика задає спосіб отримання числа, критерій – правило прийняття рішення, доказ – матеріал, який дозволяє перевірити висновок. Знімок екрана з написом «успішно» зазвичай містить менше інформації, ніж журнал із версією системи, умовами запуску, вхідними даними й фактичними результатами.

1.5 Дослідницький аспект

Наукове питання виникає, коли потрібно з’ясувати властивості самого способу оцінювання: чи справді показник відображає потрібну характеристику, які дефекти пропускає перевірка, за яких умов змінюється результат. Наприклад, можна дослідити, чи виявляє конкурентний тест подвійне бронювання краще за послідовне виконання запитів. Для відповіді потрібні однаковий об’єкт, контрольовані умови та наперед визначений критерій порівняння.

Зв’язок із лабораторною роботою №1: побудова контекстної моделі якості. Основний результат – узгоджений ланцюг «потреба – характеристика – вимога – метрика – критерій – доказ».

Питання для самоперевірки

1. Чому функціональна правильність не є достатнім доказом якості системи?
2. Чим відрізняються якість продукту та якість під час використання?
3. Які умови потрібно додати до вимоги «сервіс працює швидко»?
4. Як відрізнити метрику, критерій прийнятності та доказ?
5. Які висновки потрібно переглянути після зміни контексту використання?

Джерела: [1–3].

Тема 2. Метрики та методи вимірювання якості програмного забезпечення

План: цілі вимірювання та GQM; види метрик; шкали; надійність і валідність; нормування; інтерпретація результатів.

2.1 Ціль визначає показники

Вимірювання корисне тоді, коли допомагає відповісти на конкретне запитання. Збирання всіх доступних показників створює значний обсяг даних,

але не обов’язково покращує рішення. Підхід Goal–Question–Metric, або GQM, пов’язує ціль оцінювання із запитаннями та вимірюваними показниками [4].

Для сервісу бронювання ціль можна визначити як оцінювання стабільності обслуговування за заданого навантаження з погляду користувача. Запитання: скільки запитів завершується успішно; скільки часу триває відповідь; яка частка користувачів отримує результат у прийнятний строк. Відповідні метрики: частка успішних запитів, медіана часу, частка своєчасних успішних запитів. Ці показники доповнюють один одного, бо швидка відповідь із помилкою не виконує потребу користувача.

До плану вимірювання включають назву метрики, одиницю, формулу, джерело даних, межі вибірки, частоту вимірювання та правило інтерпретації. Окремо визначають обробку пропущених записів, повторних запитів і тайм-аутів. Зміна цих правил може істотно змінити підсумок.

2.2 Види метрик і шкали

Структурні метрики описують код і залежності: кількість розгалужень, дублювання, зв’язки між модулями. Процесні стосуються діяльності: тривалість перевірки зміни, час виправлення підтвердженого дефекту. Експлуатаційні характеризують роботу системи: затримку, помилки, використання пам’яті. Користувацькі показують виконання завдань користувачами, наприклад частку успішних бронювань без сторонньої допомоги.

Шкала	**Приклад**	**Коректне використання**
Номінальна	Тип повідомлення: помилка, попередження, рекомендація	Підрахунок частот; категорії не мають числової відстані
Порядкова	Пріоритет: низький, середній, високий	Упорядкування; відстані між рівнями не вважають однаковими
Інтервальна	Календарна дата реєстрації дефекту	Порівняння різниці дат; відношення самих дат не має змісту
Відношень	Тривалість перевірки в секундах	Різниці й відношення за однакових умов вимірювання

Не варто без обґрунтування обчислювати середню «критичність» із кодів 1, 2 і 3: числові позначення категорій не доводять однакової відстані між ними. Краще окремо подати кількість проблем кожного рівня та пояснити правила класифікації.

2.3 Надійність і валідність вимірювання

Надійність вимірювання означає його стабільність за повторення в зіставних умовах. Валідність стосується того, чи відображає показник саме потрібну властивість. Аналізатор може щоразу однаково рахувати попередження, але їхня кількість не обов’язково вимірює кількість реальних дефектів. Отже, стабільний показник може бути недостатньо валідним для обраного висновку.

Впливати на результати можуть кешування, фонові процеси, розмір даних, версії інструментів і навантаження мережі. Для порівняння часу запуску фіксують середовище та виконують повтори. Для порівняння статичних повідомлень фіксують версію коду, набір правил і спосіб підрахунку дублікатів.

2.4 Навчальний приклад обчислення

Нехай отримано час десяти запитів у мілісекундах: 120, 135, 110, 160, 125, 140, 500, 130, 150, 115. Запит тривалістю 500 мс завершився тайм-аутом, решта успішні. Сума становить 1685 мс, тому середнє дорівнює 168,5 мс. Після впорядкування середні два значення – 130 і 135; медіана дорівнює 132,5 мс.

Частка помилок = кількість невдалих запитів / кількість усіх запитів × 100 % = 1 / 10 × 100 % = 10 %.

За порога своєчасності 200 мс дев’ять запитів одночасно є успішними та своєчасними. Їхня частка становить 90 %. Середній час лише успішних запитів дорівнює 1185 / 9 ≈ 131,7 мс. Він описує успішні відповіді, але приховує очікування користувача, чий запит завершився тайм-аутом. Тому межі показника потрібно зазначати явно.

Медіана описує центральне значення й менш чутлива до окремого великого результату. Перцентиль p95 описує верхню частину розподілу, проте різні способи його обчислення на малій вибірці можуть давати різні значення. Для десяти записів краще показати всі дані та спосіб розрахунку, ніж робити сильний висновок про стабільність сервісу.

2.5 Нормування та порогові значення

Нормування приводить показники до зіставного масштабу. Проте однакова числова шкала ще не означає однакового змісту. Додавати мілісекунди до кількості дефектів не можна; навіть після нормування потрібно пояснити ваги та напрям бажаної зміни. В інтегральній оцінці добра швидкість не повинна автоматично компенсувати критичне порушення авторизації.

Пороги визначають до аналізу підсумкових результатів, спираючись на потреби, ризики та умови. Якщо поріг змінюють після пілотного вимірювання, це фіксують і перевіряють уточнений план на нових даних. Показники, які майже дублюють один одного, не варто подавати як незалежні докази.

Зв’язок із лабораторною роботою №2: розроблення плану вимірювання та його пілотна перевірка. Висновок має пояснювати виконання кожного критерію, а не лише містити числа.

Питання для самоперевірки

1. Як від цілі GQM перейти до запитань і метрик?
2. Чому повторюваний результат може бути невалідним?
3. Чим відрізняються середнє, медіана та перцентиль?
4. Як вилучення тайм-аутів вплине на висновок про якість?
5. Чому інтегральний бал може приховати критичний недолік?

Джерела: [4, 5, 11].

Тема 3. Емпіричне оцінювання методів забезпечення якості

План: питання та гіпотези; види досліджень; базовий метод; вибірка; значущість і розмір ефекту; валідність і відтворюваність.

3.1 Дослідницьке питання і гіпотеза

Емпіричне оцінювання спирається на спостережувані дані про роботу методів та інструментів. Воно потрібне, коли необхідно з’ясувати, чи допомагає

новий спосіб тестування або аналізу краще виконати конкретне завдання. Питання «який інструмент найкращий» надто широке. Коректніше запитати: «Чи виявляє конфігурація Б більше підтверджених порушень у вибраних Python-модулях, ніж конфігурація А, і скільки часу потребує перевірка її повідомлень?»

Гіпотеза формулює очікувану відповідь, яку можна перевірити. Наприклад: «Додавання перевірок граничних значень збільшує кількість виявлених контрольованих дефектів». Потрібно заздалегідь визначити дефект, спосіб його виявлення, об’єкти та критерій порівняння. Якщо зміна одночасно збільшує точність і витрати, висновок має відображати обидва результати.

Дослідницьке питання, опис об’єктів і процедура порівняння повинні узгоджуватися. Дані одного короткого прикладу дозволяють продемонструвати механізм, але не доводять переваги для всіх систем чи мов програмування.

3.2 Основні способи отримання даних

Спосіб	Що робить дослідник	Основне обмеження
Контрольований експеримент	Змінює визначений чинник і порівнює результати за узгоджених умов	Навчальне середовище може відрізнятися від реального
Дослідження випадку	Детально вивчає застосування методу в одному проєкті та його контекст	Складно відокремити всі супутні впливи
Аналіз репозиторіїв	Вивчає зміни, дефекти, обговорення та історію коду	Записи можуть бути неповними або помилково пов’язаними

Експеримент зручний для зіставлення двох наборів тестів на однакових версіях компонента. Дослідження випадку допомагає зрозуміти впровадження перевірок у команді. Аналіз репозиторіїв може показати зв’язок між частими змінами модуля та зареєстрованими дефектами. Сам по собі статистичний зв’язок ще не встановлює причинності.

3.3 Базовий метод і контроль умов

Базовий метод – це обґрунтована точка порівняння: чинна процедура, простий відтворюваний підхід або відомий метод. Навмисне послаблення базового варіанта робить висновок про перевагу недостовірним. Обидва варіанти повинні отримати зіставні дані та ресурси, а різниця налаштувань має відповідати дослідницькому питанню.

Незалежна змінна в простому експерименті – обраний метод або його конфігурація. Залежні змінні – вимірювані результати: виявлені дефекти, час, кількість непідтверджених повідомлень. Контрольовані умови – версія коду, середовище та набір вхідних даних. Якщо одночасно змінити метод, обладнання й дані, причинне пояснення результату ускладнюється.

3.4 Вибірка та повтори

Об’єкти обирають відповідно до меж висновку. Якщо досліджується код із великою кількістю розгалужень, вибірка лише лінійних функцій не відповідає питанню. Доцільно пояснити, чому включено саме ці модулі та які класи систем не представлені.

Повторні запуски вимірюють варіативність, але не створюють нові незалежні програмні проєкти. Сто запусків одного модуля залишаються спостереженнями над одним модулем. Для часу виконання можна чергувати порядок А–Б і Б–А, щоб зменшити вплив прогрівання чи зміни навантаження. Детермінований статичний аналіз незміненого коду зазвичай відтворює той самий набір повідомлень; повтор у такому разі перевіряє стабільність процедури.

3.5 Розмір ефекту та значущість

У навчальному експерименті метод А виявив 6 із 10 контрольованих дефектів, метод Б – 8 із 10. Абсолютна різниця становить два дефекти або 20 відсоткових пунктів. Відносне збільшення кількості виявлень дорівнює $(8 - 6) / 6 \times 100\ \% \approx 33{,}3\ \%$. Ці способи подання не можна змішувати.

Практична значущість залежить від того, які саме дефекти додатково виявлено та якими є витрати. Виявлення порушення контролю доступу може бути важливішим за збільшення кількості стилістичних повідомлень. Статистична значущість стосується сумісності даних із нульовою гіпотезою за припущень обраного аналізу; вона не визначає корисність результату й не є ймовірністю істинності гіпотези.

Для повного дослідження оцінюють величину ефекту та невизначеність, обирають статистичний метод з урахуванням структури даних. Для малого навчального прикладу достатньо чесного описового порівняння з усіма спостереженнями та обмеженнями; обчислення p-значення не повинно бути самоціллю.

3.6 Валідність і реплікаційний пакет

Конструктна валідність стосується відповідності показника досліджуваній властивості. Внутрішня – обґрунтованості причинного пояснення. Зовнішня – можливості поширення висновку на інші умови. Валідність висновків залежить від якості даних, достатності спостережень і коректності аналізу [5].

Для повторення зберігають код, версії інструментів, налаштування, вхідні дані, команди запуску, початкові результати та спосіб їх обчислення. Якщо є випадковість, зазначають початкове значення генератора і правила повторів. README має пояснювати, як із цих матеріалів отримати підсумкову таблицю.

Зв’язок із лабораторною роботою №3: порівняння двох конфігурацій статичного аналізу. Підрахунок повідомлень доповнюється перевіркою їхнього змісту та практичної користі.

Питання для самоперевірки

1. Які складові роблять дослідницьке питання перевірюваним?
2. Як обрати справедливий базовий метод?
3. Чому повтори одного запуску не замінюють різноманіття об’єктів?
4. Чим відрізняються 20 відсоткових пунктів і 20 відсотків?
5. Які матеріали потрібні для незалежного повторення експерименту?

Джерела: [5].

Тема 4. Статичний аналіз, архітектурна якість і технічний борг

План: статичний аналіз; структурні показники; архітектурні порушення; технічний борг; перевірка повідомлень; пріоритети виправлень.

4.1 Призначення статичного аналізу

Статичний аналіз досліджує код або його подання без виконання звичайного сценарію роботи програми. Він допомагає виявляти невизначені імена, підозрілі конструкції, порушення правил і складні залежності. Перевага – можливість регулярно перевіряти значний обсяг коду. Обмеження – неповні відомості про середовище, вхідні дані й фактичні вимоги.

ISO/IEC 5055:2021 присвячено автоматизованим показникам якості вихідного коду, пов’язаним із порушеннями архітектурних і програмних практик [6]. Застосування довільного аналізатора саме по собі не означає відповідності цьому стандарту. Потрібно встановити, які правила та показники реалізує конкретний засіб і які межі його аналізу.

Pylint перевіряє Python-код за набором правил і формує повідомлення різних категорій [7]. Radon обчислює структурні метрики, зокрема цикломатичну складність [8]. Їхні результати використовують як матеріал для аналізу, а не як остаточний висновок про якість продукту.

4.2 Складність, зв’язність і залежності

Цикломатична складність характеризує структуру потоку керування. Для простого фрагмента з одним входом і виходом її можна пояснювати через кількість рішень плюс одиниця. Якщо функція містить два прості незалежні розгалуження, базова оцінка дорівнює трьом. Конкретні інструменти можуть враховувати логічні оператори й інші конструкції по-різному; тому в дослідженні фіксують реалізацію метрики [8].

Велике значення складності сигналізує, що функцію важче аналізувати та перевіряти, але не доводить наявності дефекту. Механічне розбиття функції на багато дрібних частин може зменшити локальний показник, одночасно ускладнивши взаємодію між частинами. Потрібно оцінювати зрозумілість і збереження поведінки.

Для уникнення неоднозначності варто розрізняти внутрішню згуртованість модуля – наскільки його елементи працюють на спільне завдання – та зчеплення між модулями, тобто залежність одного від іншого. Високе зчеплення означає, що зміна інтерфейсу одного компонента може зачепити багато інших. Архітектурна якість пов’язана також із напрямом залежностей і дотриманням визначених меж.

Дублювання небезпечне, коли однакове правило доводиться синхронно виправляти в кількох місцях. Водночас схожість двох коротких фрагментів не завжди виправдовує складну спільну абстракцію. Рішення залежить від очікуваних змін і призначення коду.

4.3 Архітектурні порушення та технічний борг

Приклад архітектурного правила: інтерфейс користувача звертається до даних через сервісний шар. Якщо форма безпосередньо виконує запит до бази та дублює перевірку прав, виникає порушення узгодженої структури. Код може

працювати, але його зміни стають ризикованими: частина правил існує в кількох місцях.

Технічний борг описує рішення, що полегшує поточну роботу або залишається невиправленим, але спричиняє майбутні додаткові витрати. Для конкретного випадку слід назвати проблему, очікувані наслідки, спосіб усунення та орієнтовну трудомісткість. Не кожне попередження є технічним боргом, а автоматично оцінені «хвилини виправлення» не є точним прогнозом людської роботи.

Рефакторинг змінює внутрішню структуру зі збереженням очікуваної зовнішньої поведінки. Виправлення функціонального дефекту змінює неправильну поведінку на правильну. Їх доцільно розрізняти, щоб зрозуміти причину зміни результатів тестування.

4.4 Перевірка повідомлень

Повідомлення аналізатора – це твердження про конкретний фрагмент, яке потрібно оцінити. Спочатку читають правило, потім досліджують контекст, визначають наслідок і вирішують, чи потрібне виправлення. Хибне спрацьовування означає, що повідомлена проблема в цьому контексті не підтверджується. Непріоритетна, але справжня стилістична проблема не є хибним спрацьовуванням.

Спостереження	**Можливе пояснення**	**Додаткова перевірка**
Використано невизначене ім’я	Помилка під час виконання відповідної гілки	Виклик із даними, що досягають цієї гілки
Не використано імпорт	Зайвий елемент коду	Перевірка призначення імпорту і видалення за відсутності потрібного ефекту
Висока складність функції	Багато альтернативних шляхів	Аналіз гілок, тестів і можливості спрощення
Зменшилась кількість попереджень	Код покращено або правила вимкнено	Порівняння конфігурацій і фактичних змін

4.5 Вибір пріоритетів

Пріоритет визначають за можливим впливом, імовірністю прояву, частотою використання та витратами на усунення. Спочатку доцільно перевірити проблему, яка може зупинити основний сценарій або порушити доступ до даних. Легке виправлення стилю не повинно відволікати від небезпечної помилки лише тому, що воно швидко покращує загальний бал.

Після зміни повторюють аналіз і тести. Висновок має містити фактичну різницю: яке правило більше не порушується, чи збережено поведінку, чи не з’явилися нові проблеми. Для дослідження зв’язку метрик із дефектами потрібні дані про підтверджені дефекти та зміни; самого звіту аналізатора недостатньо.

Зв’язок із лабораторною роботою №4: аналіз невеликого компонента, вибір одного обґрунтованого виправлення та порівняння стану до і після зміни.

Питання для самоперевірки

1. Які проблеми статичний аналіз виявляє без запуску програми?
2. Чому складність не дорівнює кількості дефектів?
3. Чим хибне спрацьовування відрізняється від непріоритетного порушення?
4. Як відрізнити рефакторинг від виправлення функціональної помилки?
5. Якими доказами підтвердити користь виконаної зміни?

Джерела: [6–8].

Тема 5. Сучасні методи тестування та оцінювання ефективності тестів

План: ризики та тестовий оракул; тестування за моделями, прикладами й властивостями; метаморфні перевірки; мутаційний аналіз; покриття та порівняння тестів.

5.1 Що доводить виконаний тест

Тест зіставляє фактичну поведінку з очікуваною за певних умов. Успішний тест означає, що конкретна перевірка не виявила порушення. Він не доводить правильності для всіх можливих входів. Тестовий оракул – правило або джерело, за яким визначають правильність результату: вимога, відома відповідь, незалежна реалізація чи властивість.

Наприклад, компонент clamp(x, low, high) має повертати low, коли x нижче інтервалу; high, коли x вище; саме x, коли воно всередині. Передумова – low ≤ high. Для значень x = −1, low = 0, high = 100 очікуваний результат дорівнює 0. Специфікація визначає оракул для цього прикладу. Окремо необхідно встановити поведінку за некоректних типів і перевернутого інтервалу.

Ризик-орієнтоване тестування спрямовує зусилля на сценарії з істотними наслідками відмов. У сервісі бронювання порушення авторизації або подвійний запис можуть мати вищий пріоритет, ніж неточне вирівнювання тексту. Визначення пріоритету не означає вилучення інших перевірок; воно допомагає розподілити обмежений час.

5.2 Приклади, властивості та моделі

Тестування на основі прикладів використовує вибрані входи й конкретні очікувані відповіді. Для clamp потрібно охопити значення нижче, всередині й вище інтервалу, його межі та некоректні дані. Один «звичайний» вхід не представляє всі класи поведінки.

Тестування на основі властивостей задає загальне твердження та перевіряє його на автоматично згенерованих входах. Hypothesis підтримує такий підхід для Python і допомагає знайти компактний приклад, на якому властивість порушується [9]. Генератор повинен відповідати області, для якої твердження справедливе.

Для clamp можна визначити три властивості: результат належить інтервалу; повторне обмеження не змінює результат; значення всередині інтервалу повертається незмінним. Перша властивість сама по собі слабка: неправильна реалізація, яка завжди повертає low, теж її виконує. Поєднання властивостей посилює перевірку, але не звільняє від аналізу специфікації.

Модельно-орієнтоване тестування виводить сценарії зі спрощеної моделі поведінки. Наприклад, бронювання проходить стани «створено», «підтверджено», «скасовано». Модель визначає дозволені переходи та заборонені дії. Якщо модель неповна або помилкова, згенеровані тести успадковують її обмеження.

5.3 Метаморфне та інші види тестування

Метаморфне тестування корисне, коли точна відповідь для складного входу невідома, але відомий зв’язок між кількома виконаннями. Для сортування перестановка вхідного списку не повинна змінювати відсортований результат. Така перевірка не потребує ручного запису правильної відповіді для кожного списку, проте залежить від коректності обраного відношення.

Контрактне тестування перевіряє узгоджені очікування між компонентами: формат запиту, обов’язкові поля, допустимі відповіді. Fuzzing подає різноманітні, зокрема нетипові, входи для виявлення аварій, зависань та інших порушень. Формальна верифікація перевіряє властивості відносно математично визначеної моделі або специфікації. Її висновок залежить від прийнятих припущень і відповідності моделі реальній системі.

Ці підходи доповнюють один одного. Контракт API не доводить правильності бізнес-правил, а відсутність аварій під час fuzzing не доводить відсутності витоку даних. Метод обирають за видом ризику та властивістю, яку потрібно перевірити.

5.4 Мутаційне тестування

Мутаційне тестування оцінює здатність набору тестів помічати невеликі контрольовані зміни програми [10]. Змінена версія називається мутантом. Наприклад, return low замінюють на return low + 1. Якщо тести успішні на початковій програмі, але виявляють порушення на зміненій, мутант вважається виявленим.

Для навчального підрахунку застосуємо $MS = K / (M - E) \times 100\ \%$, де K – кількість виявлених мутантів, M – кількість коректно створених мутантів, E – кількість доведено еквівалентних. Формула застосовна, коли $M > E$. Мутанти, які не вдалося коректно побудувати або запустити через технічну помилку, потребують окремого обліку.

Якщо з п’яти коректних нееквівалентних мутантів тести виявили три, MS дорівнює 60 %. Якщо інший набір виявив чотири, його результат – 80 %. Різниця становить 20 відсоткових пунктів. Висновок стосується саме вибраних змін і входів, а не загальної частки всіх можливих дефектів у програмі.

Еквівалентний мутант не змінює спостережувану поведінку в установленій області. Те, що мутант залишився невиявленим, ще не доводить еквівалентності: можливо, бракує потрібного входу або перевірка результату надто слабка. Припущення про еквівалентність необхідно обґрунтувати.

5.5 Як порівнювати тести

Показник	Що показує	Чого самостійно не доводить
Покриття рядків або гілок	Які елементи виконувалися	Що результат їх виконання перевірено правильно
Кількість виявлених контрольованих дефектів	Чутливість до визначеного набору порушень	Перевагу на всіх реальних дефектах
Мутаційний показник	Частку виявлених змін серед урахованих мутантів	Загальний рівень якості продукту
Час і витрати підтримки	Ресурсну ціну перевірки	Достатність перевіреної поведінки

Порівнюють набори на тих самих версіях і мутантах. Час вимірюють за однакових умов; швидке аварійне завершення не трактують як перевагу. Для випадкових генераторів враховують варіативність і зберігають налаштування. Кількість тестових функцій не дорівнює кількості згенерованих прикладів.

Зв’язок із лабораторною роботою №5: порівняння тестів на прикладах і властивостях за допомогою однакового набору контрольованих змін.

Питання для самоперевірки

1. Що може бути тестовим оракулом?
2. Чому властивості «результат у межах» недостатньо для clamp?
3. Який зв’язок перевіряє метаморфне тестування?
4. Чому невиявлений мутант не можна автоматично називати еквівалентним?
5. Чому високе покриття не гарантує ефективності тестів?

Джерела: [9, 10, 15].

Тема 6. Безперервне забезпечення якості, DevSecOps та експлуатаційний контроль

План: перевірки в CI/CD; умови прийняття змін; перевірки безпеки; ASVS і SAMM; телеметрія та SLI/SLO.

6.1 Перевірки протягом життєвого циклу

Безперервне забезпечення якості означає систематичне отримання й використання доказів під час зміни та експлуатації програмного забезпечення. Невеликі перевірювані зміни дають змогу раніше помічати дефекти. Водночас автоматизація корисна лише тоді, коли перевірки відповідають ризикам і результати впливають на рішення.

Continuous Integration, CI, передбачає регулярну інтеграцію змін з автоматизованими перевірками. Continuous Delivery підтримує готовність до випуску, а Continuous Deployment передбачає автоматичне розгортання змін, які пройшли визначені умови. Для навчального конвеєра достатньо автоматичного запуску тестів і аналізу без розгортання продукту.

У GitLab конвеєр складається із завдань, що запускаються відповідно до конфігурації; їх виконує Runner. Завдання можуть зберігати артефакти –

журнали, результати тестів та інші файли [12]. Відсутність доступного Runner означає, що перевірка ще не відбулася; це не є доказом дефекту програми.

6.2 Побудова конвеєра й умови прийняття

Навчальна послідовність може включати підготовку середовища, тести, статичний аналіз і одну перевірку безпеки. На кожному етапі визначають команду, очікуваний результат, умову невдачі та матеріали для перегляду. Перевірки мають завершуватися зрозумілим статусом, а журнали – дозволяти з'ясувати причину.

Quality gate – це правило, яке дозволяє або блокує перехід, наприклад прийняття зміни до основної гілки. Правило «усі обов'язкові тести успішні» має сенс лише тоді, коли ці тести справді запускаються й їхній невдалий результат не ігнорується. Потрібно перевірити також налаштування прийняття змін у репозиторії.

Перевірка	**Приклад умови прийняття**	**Матеріал для аналізу**
Функціональні тести	Немає невдалих обов'язкових тестів	Список тестів і повідомлення про порушення
Статичний аналіз	Немає порушень узгодженого блокувального набору правил	Правило, файл, місце та пояснення
Перевірка залежностей	Немає неприйнятих за політикою підтверджених ризиків	Версія залежності, повідомлення та рішення щодо нього
Пошук секретів	Не виявлено незахищених облікових даних у перевіреній області	Результат перевірки без розкриття самого секрету

Умови не слід визначати як «відсутність будь-яких повідомлень» без урахування їхнього змісту. Для наявних проблем можна створити контрольований перелік із відповідальними та строками перегляду, водночас блокуючи нові критичні порушення. Виняток має бути явним рішенням, а не прихованим вимкненням перевірки.

6.3 DevSecOps і межі перевірок безпеки

DevSecOps інтегрує безпеку в розроблення та експлуатацію. Потрібні вимоги, перевірки коду й залежностей, керування доступом і реагування на результати. Один сканер не охоплює всі ці завдання. Наприклад, пошук текстового маркера може продемонструвати роботу блокувального правила, але не замінює повноцінного виявлення секретів.

Статичні перевірки можуть знайти підозрілий потік даних у коді, динамічні – проблему в запущеному застосунку, а перевірки залежностей – відому вразливість компонента. Результат оцінюють у контексті: чи використовується вразлива функція, які дані доступні, чи є компенсувальні заходи. Висновок «інструмент нічого не знайшов» не рівнозначний висновку «система безпечна».

OWASP ASVS визначає вимоги для перевірки технічних засобів безпеки застосунків; у матеріалі використано редакцію 5.0.0. OWASP SAMM допомагає оцінювати й розвивати процеси забезпечення безпеки [13, 14]. Для перевірки вебсервісу можна обрати доречні вимоги авторизації з ASVS; для аналізу

організації автоматизованого тестування – практику Security Testing із SAMM. Наявність однієї перевірки не означає виконання всього стандарту або досягнення певної зрілості.

6.4 Телеметрія та показники експлуатації

Телеметрія включає дані про виконання: метрики, журнали подій і трасування запитів. Вона допомагає виявляти проблеми, які не проявилися до випуску. Важливо пов’язати спостереження з версією, часом і контекстом, а також не зберігати зайві чутливі дані в журналах.

SLI – показник рівня обслуговування, наприклад частка успішних запитів. SLO – цільове значення цього показника за визначений період і для визначеної сукупності запитів. SLA – домовленість про рівень обслуговування з установленими умовами й наслідками. Ці поняття не слід ототожнювати [11].

Умовний SLO: не менше 99 % із врахованих запитів за місяць успішно завершуються. Якщо враховано 10 000 запитів, допустима кількість невдалих за таким критерієм – 100. Це бюджет помилок для запитів; його не можна автоматично перетворювати на хвилини простою, бо інтенсивність звернень нерівномірна.

6.5 Перевірка працездатності самої процедури

Корисна навчальна перевірка має три стани: справний компонент проходить конвеєр; контрольована помилка призводить до передбаченого блокування; після усунення помилки конвеєр знову успішний. Така послідовність показує, що правило реагує на потрібну зміну, а успішний результат не є наслідком пропуску завдання.

Зв’язок із лабораторною роботою №6: налаштування автоматизованих перевірок і демонстрація їхньої реакції на контрольовану проблему.

Питання для самоперевірки

1. Чим відрізняються запуск перевірки та блокування неприйнятної зміни?
2. Які артефакти потрібні для аналізу невдалого конвеєра?
3. Чим ASVS відрізняється від SAMM?
4. Як розрізнити SLI, SLO та SLA?
5. Навіщо перевіряти послідовність «успіх – проблема – усунення»?

Джерела: [11–14].

Тема 7. Особливості оцінювання якості інтелектуальних і розподілених систем

План: дані та моделі машинного навчання; робастність і дрейф; справедливість і пояснюваність; розподілені системи; контрольовані сценарії; межі висновків.

7.1 Чому потрібні спеціальні способи оцінювання

Для інтелектуальної системи поведінка залежить не лише від коду, а й від даних, процедури навчання та версії моделі. Для розподіленої – від взаємодії компонентів, затримок мережі та часткових відмов. Тому перевірки окремої функції не охоплюють усіх істотних властивостей таких систем.

Спочатку обирають сценарій і характеристику. Для прогнозу часу виконання заявки важлива помилка прогнозування; для сервісу бронювання із зовнішньою базою – поведінка за недоступності залежності. Один навчальний експеримент доцільно присвятити визначеній властивості одного класу систем, зберігаючи зрозумілі умови порівняння.

7.2 Якість даних і моделі

Якість даних включає коректність значень і міток, повноту, узгодженість та придатність для обраного завдання. Велика кількість записів не компенсує систематичної помилки в розмітці. Якщо певний тип заявок майже не представлений, середня оцінка на всій вибірці може приховувати низьку якість саме для цього типу.

Навчальні дані використовують для побудови моделі, валідаційні – для вибору налаштувань, тестові – для підсумкової оцінки. Повторне підлаштування за результатами тестової вибірки робить її частиною процесу вибору. Витік даних виникає, зокрема, коли до прогнозу потрапляє інформація, яка в реальному сценарії ще невідома.

Умовний класифікатор має 100 прикладів, серед яких лише 5 належать до важливого рідкісного класу. Передбачення «звичайний» для всіх прикладів дає 95 % правильних відповідей, але не виявляє жодного рідкісного випадку. Отже, загальна accuracy недостатня. Для позитивного класу окремо розглядають precision = TP / (TP + FP) та recall = TP / (TP + FN), якщо знаменники ненульові. TP – правильно виявлені позитивні випадки, FP – хибні позитивні, FN – пропущені позитивні.

7.3 Робастність, справедливість, пояснюваність і дрейф

Робастність характеризує стійкість результатів до допустимих змін даних або умов. Наприклад, перевіряють вплив невеликого шуму чи частини пропущених значень. Потрібно пояснити, чому збурення є реалістичним і чи не змінює воно саме правильне значення відповіді.

Справедливість оцінюють з урахуванням контексту та наслідків для різних груп. Порівнювати лише загальну точність недостатньо, якщо помилки нерівномірно розподілені. Пояснюваність допомагає зрозуміти підстави результату, але наявність пояснення сама по собі не доводить його правильності. NIST AI RMF 1.0 розглядає надійність, пояснюваність і керування шкідливими упередженнями як взаємопов’язані аспекти довіри до систем ШІ [16].

Дрейф даних означає зміну розподілу входів, а дрейф концепції – зміну зв’язку між входами та потрібною відповіддю. Наприклад, новий тип заявок змінює склад даних, а зміна правил їх обробки може змінити залежність між ознаками та часом виконання. Виявлена зміна розподілу ще не доводить погіршення результатів: потрібна перевірка показників на актуальних даних.

7.4 Властивості розподілених систем

У розподіленій системі одна частина може залишатися доступною, коли інша не відповідає. Важливо перевірити тайм-аути, повторні запити, поведінку за застарілих даних і відновлення. Узгодженість даних описує гарантії

спостереження результатів операцій; доступність – можливість отримати потрібне обслуговування за визначених умов; затримка – час відповіді; відмовостійкість – здатність продовжувати допустиму роботу за певних відмов.

Повтор запиту може допомогти подолати короткочасний збій, але створити подвійне бронювання, якщо операція не захищена від повторного виконання. Ідемпотентність означає, що повторення тієї самої операції має той самий зовнішній ефект, що й одноразове виконання. Для операцій створення це часто потребує спеціального ключа або іншого механізму розпізнавання повтору.

Резервний компонент може підвищити доступність, але додати затримку, вартість і складність узгодження. Тому рішення порівнюють за кількома характеристиками та явно зазначають компроміс.

7.5 Навчальний сценарій порівняння

Нехай локальна модель імітує 100 звернень до сервісу. У базовому сценарії всі звернення успішні; за зміни умов кожне п’яте завершується відмовою залежності. Частка успішних звернень змінюється зі 100 % до 80 %. Додатково вимірюють частку успішних відповідей, які надійшли до встановленого строку.

Сценарій	**Що змінено**	**Що потрібно зафіксувати**
Базовий	Залежність доступна	Кількість звернень, успіхи, час
Часткова відмова	Задана частина звернень неуспішна	Правило відмови, тайм-аут і результат
Відновлення або інше правило повтору	Змінено один механізм обробки	Успіхи, затримку, додаткові звернення

Результат локальної моделі підтверджує поведінку заданого сценарію. Він не є вимірюванням реальної хмарної системи. Для реального сервісу потрібні окремі експерименти з фактичним навантаженням, середовищем і залежностями. Для стохастичних моделей виконують повтори; незмінний детермінований сценарій має відтворювати ті самі числа.

Зв’язок із лабораторною роботою №7: оцінювання однієї-двох характеристик у базових і змінених умовах із поясненням меж застосовності результату.

Питання для самоперевірки

1. Чому висока accuracy може приховати пропуск важливих випадків?
2. Як витік даних спотворює оцінювання моделі?
3. Чим дрейф даних відрізняється від дрейфу концепції?
4. Які ризики створює повторне надсилання запиту?
5. Яких висновків не можна робити за результатами локальної імітації?

Джерела: [1, 11, 16, 17].

Тема 8. Розроблення та емпірична перевірка вдосконалення методу забезпечення якості

План: проблема і запропонована зміна; базовий підхід; критерії; порівняльний експеримент; чутливість і внесок компонентів; інтерпретація та науковий звіт.

8.1 Від спостереження до вдосконалення

Вдосконалення починається з конкретного обмеження наявного методу. Приклади: тести не охоплюють межі інтервалу; аналізатор створює багато повторних повідомлень; показник середнього часу приховує невдалі запити. Формулювання проблеми має описувати наслідок: які помилки залишаються непоміченими або яке рішення може бути хибним.

Зміна повинна бути достатньо визначеною для повторення. Замість «покращити тестування» варто написати «додати перевірки значень low − 1, low, high та high + 1 і окрему перевірку інтервалу low = high». Таку пропозицію можна реалізувати, порівняти з початковим набором і проаналізувати її внесок.

Навчальне локальне вдосконалення показує здатність провести дослідження. Щоб заявляти наукову новизну для дисертації або публікації, потрібно додатково зіставити пропозицію з відомими підходами та пояснити, у чому саме полягає нове знання. Сам факт використання нового для здобувача інструмента не є доказом новизни.

8.2 План перевірки

Спочатку фіксують базовий варіант А та змінений варіант Б. Обирають спільні об’єкти, критерії й умови. Основний показник відповідає проблемі, а додаткові відображають витрати та небажані наслідки. Наприклад, до кількості виявлених дефектів додають час виконання і складність підтримки тестів.

Матеріали, на яких налаштовували метод, бажано відокремити від підсумкової перевірки. Якщо тести створено спеціально для п’яти відомих мутантів, успіх на цих самих мутантах може відображати пристосування до прикладів. Додатковий набір контрольованих змін допомагає перевірити, чи зберігається корисність за межами налаштування.

До запуску визначають, що вважатиметься покращенням. Наприклад: більше виявлених нееквівалентних мутантів за прийнятного збільшення часу. Якщо показники суперечать один одному, висновок повинен пояснювати компроміс, а не вибирати лише вигідний результат.

8.3 Приклад порівняльного експерименту

Умовний набір А містить тести лише для значень усередині інтервалу та далеко поза ним. Варіант Б додає перевірки меж. На трьох нових контрольованих змінах А виявляє одну, Б – дві. На початковій справній реалізації обидва набори мають завершитися успішно. Інакше потрібно спочатку усунути помилки самих тестів.

Показник	Варіант А	Варіант Б	Інтерпретація
Виявлені нові мутанти	1 із 3	2 із 3	Додатково виявлено одну контрольовану зміну
Частка виявлення	33,3 %	66,7 %	Зростання приблизно на 33,3 відсоткового пункта
Медіана часу виконання	0,20 с	0,24 с	Збільшення на 0,04 с, або 20 %

Це навчальні числа. За ними можна сказати, що додані перевірки виявили ще одну зміну в цьому наборі. Не можна стверджувати, що метод «удвічі підвищує якість будь-якого ПЗ». Потрібно також пояснити природу невиявленої зміни й оцінити, чи прийнятні додаткові витрати.

Для часових вимірювань наводять повтори та умови. Якщо різниця менша за звичайні коливання часу в середовищі, висновок про прискорення чи уповільнення залишається невизначеним. Збільшення кількості знаків після коми не усуває цієї невизначеності.

8.4 Чутливість і внесок окремого компонента

Аналіз чутливості перевіряє, як змінюється висновок за зміни параметра: порога часу, кількості генерованих входів або обсягу даних. Якщо перевага існує лише для одного довільно підібраного порога, потрібно пояснити причину та обмеження.

Аналіз внеску компонента порівнює повний варіант із варіантом без однієї складової. Наприклад, окремо додають перевірки меж і перевірки некоректних типів, а потім їх поєднання. Це дозволяє з’ясувати, яка частина забезпечує додаткові виявлення. Якщо одночасно змінити все, внесок окремого рішення буде невідомим.

У невеликому навчальному дослідженні достатньо однієї чіткої зміни та контрольованого порівняння. Розширення кількості варіантів має відповідати дослідницькому питанню, а не ускладнювати роботу без потреби.

8.5 Позитивні, негативні та невизначені результати

Позитивний результат означає виконання наперед визначеного критерію в досліджених умовах. Негативний – відсутність очікуваної переваги або погіршення. Невизначений – недостатність даних чи надмірну мінливість для переконливого висновку. Усі ці результати мають дослідницьку цінність, якщо процедура прозора.

Не потрібно приховувати приклади, на яких варіант Б працює гірше. Вони допомагають визначити межі методу та наступний напрям роботи. Заміна критерію після отримання результатів потребує явного пояснення й нової перевірки; вона не повинна маскувати невдалу гіпотезу.

8.6 Структура стислого наукового викладу

Опис дослідження включає проблему й мету, базовий метод, запропоновану зміну, дані та процедуру, результати, їхню інтерпретацію й обмеження. Читач повинен бачити, що саме змінили, за яким критерієм порівнювали та які докази підтримують висновок. Початкові дані й команди відтворення зберігають разом із матеріалами.

Для навчальної роботи це може бути електронний README із таблицею та посиланнями на результати. Формат подання не зменшує вимог до обґрунтованості. Головне – можливість простежити перехід від даних до висновку та повторити ключові дії.

Зв’язок із лабораторною роботою №8: запропонувати одне вдосконалення, перевірити його порівняно з базовим підходом і подати стислий електронний виклад результатів.

Питання для самоперевірки

1. Як перетворити загальну ідею покращення на перевірювану зміну?
2. Навіщо відокремлювати дані налаштування від підсумкової перевірки?
3. Чим аналіз чутливості відрізняється від аналізу внеску компонента?
4. Як сформулювати висновок за відсутності переваги?
5. Чим локальне вдосконалення відрізняється від доведеної наукової новизни?

Джерела: [5, 10].

Основні терміни

Термін	Значення в межах дисципліни
Характеристика якості	Властивість, за якою описують та оцінюють якість
Метрика	Визначений спосіб кількісного опису властивості
Критерій прийнятності	Правило рішення про виконання вимоги
Доказ	Дані або матеріал, що підтримує перевірюваний висновок
GQM	Зв’язок цілі вимірювання, запитань і метрик
Базовий метод	Обґрунтований варіант для порівняння
Валідність	Обґрунтованість вимірювання, пояснення або висновку в установлених межах
Тестовий оракул	Джерело або правило визначення правильності результату
Мутант	Версія програми з контрольованою зміною для оцінювання тестів
Quality gate	Умова дозволу або блокування переходу чи зміни
SLI та SLO	Показник рівня обслуговування та цільове значення для нього
Реплікаційний пакет	Матеріали, потрібні для повторення дослідження

Узагальнювальні питання

1. Як пов’язані контекст використання, модель якості, метрики та експеримент?
2. Які різні докази дають статичний аналіз, тестування й експлуатаційні спостереження?
3. Як показати користь методу без перебільшення результатів малого експерименту?
4. Яку одну процедуру забезпечення якості можна дослідити у власній дисертаційній роботі?

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 25002:2024. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) – Quality model overview and usage. URL: https://www.iso.org/standard/78175.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. ISO/IEC 5055:2021. Information technology – Software measurement – Software quality measurement – Automated source code quality measures. URL: https://www.iso.org/standard/80623.html.
7. Pylint. Running Pylint. Офіційна документація. URL: https://pylint.readthedocs.io/en/stable/user_guide/usage/run.html.
8. Radon. Introduction to Code Metrics. Офіційна документація. URL: https://radon.readthedocs.io/en/latest/intro.html.
9. Hypothesis. Офіційна документація бібліотеки тестування на основі властивостей. URL: https://hypothesis.readthedocs.io/en/latest/.
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. 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. Розділ Implementing SLOs. URL: https://sre.google/workbook/implementing-slos/.
12. GitLab. CI/CD pipelines. Офіційна документація. URL: https://docs.gitlab.com/ci/pipelines/.
13. OWASP. Application Security Verification Standard. Version 5.0.0, 2025. URL: https://github.com/OWASP/ASVS/releases/tag/v5.0.0.
14. OWASP SAMM. Security Testing. URL: https://owaspsamm.org/model/verification/security-testing/.
15. pytest. How to invoke pytest. Офіційна документація. URL: https://docs.pytest.org/en/stable/how-to/usage.html.
16. Tabassi E. Artificial Intelligence Risk Management Framework (AI RMF 1.0). NIST AI 100-1. 2023. DOI: https://doi.org/10.6028/NIST.AI.100-1.
17. scikit-learn. Common pitfalls and recommended practices. Офіційна документація. URL: https://scikit-learn.org/stable/common_pitfalls.html.

Навчально-методичне видання

МЕТОДИ ЗАБЕЗПЕЧЕННЯ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Конспект лекцій
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