Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження ефективності code coverage в автоматизованому тестуванні веб-застосунків
(тема)

Виконав:
здобувач 2 року навчання
групи ІПЗм-23-2

Данило КУНЧЕНКО
(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-наукова

Керівник проф. Наталя ЛЄСНА
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Кунченко Данилові Вячеславовичу
(прізвище, ім’я, по батькові)

1. Тема роботи «Дослідження ефективності code coverage в автоматизованому тестуванні веб-застосунків»

Затверджена наказом по університету від 15.04. 2025р. № 290 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 14.06.2025

3. Вихідні дані до роботи рекомендації по використанню та підключенню Code coverage до веб-застосунку. Рекомендації щодо інтеграції Code coverage на CI.

4. Перелік питань, що потрібно опрацювати в роботі

Чи впливає високий рівень покриття коду на якість продукту? Як краще підключати Code coverage на CI? Які плюси та мінусу використовування Code coverage на проєкті?

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	21 січня 2025р.	*виконано*
2	Аналіз предметної галузі і постановка задачі	4 лютого 2025 р.	*виконано*
3	Назва за розділами теоретичного і практичного дослідження	25 лютого 2025 р.	*виконано*
4	Назва за розділами теоретичного і практичного дослідження	20 березня 2025 р.	*виконано*
5	Підготовка до апробації результатів дослідження. Публікація матеріалів	10 квітня 2025 р.	*виконано*
6	Назва за розділами теоретичного і практичного дослідження	25 квітня 2025 р.	*виконано*
7	Підготовка пояснювальної записки	10 травня 2025 р.	*виконано*
8	Підготовка презентації та доповіді	15 травня 2025 р.	*виконано*
9	Перевірка на плагіат	18 травня 2025 р.	*виконано*
10	Нормоконтроль	22 травня 2025 р.	*виконано*
11	Рецензування	28 травня 2025 р.	*виконано*
12	Попередній захист	29 травня 2025 р.	*виконано*
13	Занесення диплома в електронний архів	2 червня 2025 р.	*виконано*
14	Допуск до захисту у зав. кафедри	5 червня 2025 р.	*виконано*

Дата видачі завдання 20 січня 2025р.

Студент (ка) _____ Данило КУНЧЕКО
(підпис)

Керівник роботи _____ проф. Наталя ЛЄСНА
(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить: 70с., 13 рис., 5 табл., 14 джерел.

АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ, ВЕБ-ЗАСТОСУНОК, МЕТРИКИ ПОКРИТТЯ КОДУ, ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, CODE COVERAGE, CODECEPTION, CONTINUOS INTEGRATION, DOCKER, PHP, XDEBUG.

Метою даної роботи є комплексний аналіз впливу покриття коду на якість веб-застосунків. Дослідження спрямоване на визначення взаємозв'язку між рівнем тестового покриття та ефективністю програмного забезпечення, виявлення методологічних особливостей автоматизованого тестування.

Об'єктом дослідження є процеси забезпечення якості програмного забезпечення через автоматизоване тестування. Основна увага приділяється методологіям, інструментам та стратегіям тестування, які дозволяють підвищити надійність та продуктивність веб-додатків. Предметом дослідження слугують метрики покриття коду, їх типологія, взаємозв'язок з якістю тестування та потенціал щодо виявлення потенційних вразливостей у програмному забезпеченні. Аналіз буде проведено з використанням PHP, фреймворку Codeception та інструменту Xdebug.

В рамках практичної частини дослідження було успішно реалізовано інтеграцію інструментів вимірювання покриття коду (Codeception та Xdebug) у процес розробки веб-застосунку Article Hub. Проведене дослідження показало, що високий рівень покриття коду позитивно впливає на якість програмного продукту, проте цей вплив має певні обмеження:

а) покриття коду є лише одним із багатьох показників якості програмного забезпечення. Високий відсоток покриття не гарантує відсутність помилок та не забезпечує повну якість продукту;
б) важливо враховувати й інші аспекти якості програмного забезпечення:
    1) продуктивність та оптимізація;

2) безпека та захищеність даних;
3) зручність використання (usability);
4) масштабованість;
5) підтримуваність коду;
6) документованість;

в) реалізована інтеграція з системами безперервної інтеграції (CI) дозволила автоматизувати процес контролю якості, але потребує правильного налаштування та регулярного моніторингу.

Практична значимість роботи полягає в розробці конкретних рекомендацій щодо налаштування інструментів вимірювання покриття коду, інтерпретації метрик покриття, впровадження процесів контролю якості в CI, та встановлення оптимальних порогових значень покриття. Результати дослідження можуть бути використані для покращення процесів забезпечення якості програмного забезпечення в компаніях, що займаються розробкою веб-додатків, а також для навчання студентів і підвищення кваліфікації тестувальників.

AUTOMATION TESTING, CODE COVERAGE, PHP, SOFTWARE QUALITY, WEB APPLICATION, CONTINUOS INTEGRATION, CODECEPTION, XDEBUG, DOCKER, CODE COVERAGE METRICS.

The aim of this work is to comprehensively analyze the impact of code coverage on the quality of web applications. The research is directed towards determining the relationship between the level of test coverage and software efficiency, as well as identifying methodological features of automated testing. The object of the study is the processes of software quality assurance through automated testing. The main focus is on testing methodologies, tools, and strategies that enhance the reliability and performance of web applications. The subject of the study includes code coverage metrics, their typology, the relationship with test quality, and the potential for identifying vulnerabilities in software. The analysis will be conducted using PHP, the Codeception framework, and the Xdebug tool.

In the practical part of the research, the integration of code coverage measurement tools (Codeception and Xdebug) into the development process of the Article Hub web application was successfully implemented. The research showed that a high level of code coverage positively affects the quality of the software product, but this influence has certain limitations:

а) code coverage is only one of many indicators of software quality. A high percentage of coverage does not guarantee the absence of errors and does not ensure complete product quality;

б) it is important to consider other aspects of software quality:

1) performance and optimization;
2) security and data protection;
3) usability;
4) scalability;
5) code maintainability;
6) documentation;

в) the implemented integration with continuous integration (CI) systems allowed for the automation of the quality control process, but it requires proper configuration and regular monitoring.

The practical significance of this work lies in the development of specific recommendations for configuring code coverage measurement tools, interpreting coverage metrics, implementing quality control processes in CI, and setting optimal threshold values for coverage. The results of the research can be used to improve software quality assurance processes in companies developing web applications, as well as for training students and enhancing the qualifications of testers.

Завідувачу кафедри
ПІ_____
(скорочена назва кафедри)
проф. Кирилу СМЕЛЯКОВУ_
(вчене звання, сласне ім'я, прізвище)

ЗАЯВА
щодо самостійності виконання кваліфікаційної роботи та можливості її публікації (та/або публікації анотації кваліфікаційної роботи) в електронному архіві відкритого доступу ElAr KhNURE

Я, Кунченко Данило Вячеславович, студент(ка) гр. ІПЗм-23-2, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження ефективності code coverage в автоматизованому тестуванні веб-застосунків», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений(на) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

Дата Підпис

ПЕРЕЛІК СКОРОЧЕНЬ

CI – Continuous Integration

CD – Continuous delivery

ЗМІСТ

ВСТУП

У сучасному світі веб-застосунки займають значну роль у багатьох сферах життя, від бізнесу та освіти до розваг і комунікацій. З кожним днем вимоги до якості та надійності програмного забезпечення зростають, що вимагає застосування більш ефективних методів тестування. Одним з ключових аспектів забезпечення високої якості програмного забезпечення є рівень покриття коду (code coverage) в автоматизованому тестуванні. Code coverage дозволяє оцінити, наскільки добре тестами покрито програмний код, виявити не тестовані ділянки та потенційні дефекти. Однак, високий рівень покриття не завжди гарантує відсутність помилок або повну перевірку функціональності. Тому актуальним є дослідження ефективності різних метрик покриття коду та їх впливу на якість веб-застосунків.

Мета дослідження полягає у комплексному аналізі впливу покриття коду на якість веб-застосунків. Дослідження спрямоване на визначення взаємозв'язку між рівнем тестового покриття та ефективністю програмного забезпечення, виявлення методологічних особливостей автоматизованого тестування.

Об'єктом дослідження є процеси забезпечення якості програмного забезпечення через автоматизоване тестування. Основна увага приділяється методологіям, інструментам та стратегіям тестування, які дозволяють підвищити надійність та продуктивність веб-додатків.

Предметом дослідження слугують метрики покриття коду, їх типологія, взаємозв'язок з якістю тестування та потенціал щодо виявлення потенційних вразливостей у програмному забезпеченні. Аналіз буде проведено з використанням PHP, фреймворку Codeception та інструменту Xdebug.

Очікувані результати включають кількісну оцінку впливу code coverage на виявлення програмних дефектів, розробку практичних рекомендацій з оптимізації тестових процесів та визначення оптимальних стратегій забезпечення якості веб-застосунків.

Практична значимість дослідження полягає у формуванні науково обґрунтованих підходів до автоматизованого тестування, які дозволять підвищити ефективність розробки та супроводження програмного забезпечення.

У результаті проведених досліджень були визначені найбільш ефективні методи та інструменти для оцінки покриття коду в автоматизованому тестуванні веб-застосунків. Встановлено, що певні метрики покриття коду мають більший вплив на якість програмного забезпечення. Виявлені основні проблеми, пов'язані з досягненням високого покриття коду, і розроблені рекомендації щодо їх подолання. Результати дослідження можуть бути використані для покращення процесів забезпечення якості програмного забезпечення в компаніях, що займаються розробкою веб-додатків, а також для навчання студентів і підвищення кваліфікації тестувальників.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ І ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної галузі

Аналіз предметної галузі охоплює дослідження специфіки автоматизованого тестування веб-застосунків, зокрема методів та інструментів оцінки покриття коду. Важливо зрозуміти існуючі підходи, обмеження, тенденції та перспективи в цій галузі, щоб ефективно оцінити значення покриття коду для забезпечення якості програмного забезпечення.

Сучасний ландшафт автоматизованого тестування характеризується високою динамічністю та постійною еволюцією методологічних та технологічних підходів. Це пов'язано з безперервним ускладненням архітектури веб-застосунків, розширенням функціональних вимог та необхідністю забезпечення високої надійності програмних продуктів в умовах жорсткої ринкової конкуренції.

Принципового значення набуває міждисциплінарний підхід до дослідження проблематики покриття коду, що передбачає інтеграцію методологій з різних галузей. Такий комплексний підхід дозволяє не лише оцінити поточний стан тестового забезпечення, але й розробити прогностичні моделі підвищення якості програмних систем.

Особливої уваги потребує дослідження взаємозв'язку між формальними показниками покриття коду та реальною якістю програмного забезпечення. Статистичний аналіз великої кількості проєктів демонструє, що високий відсоток покриття не завжди корелює з мінімізацією дефектів та забезпеченням стабільності системи.

1.2 Специфіка автоматизованого тестування веб-застосунків

Автоматизоване тестування веб-застосунків є ключовим компонентом сучасного розробницького процесу. Воно дозволяє швидко і ефективно виявляти помилки, забезпечувати стабільність та високу якість продукту. Основними видами тестування є функціональне, інтеграційне, API-тестування, та юніт-тестування. Кожен з цих видів тестування виконує специфічні завдання та використовує різні підходи до оцінки покриття коду.

Архітектурна складність сучасних веб-застосунків висуває принципово нові вимоги до методологій автоматизованого тестування. Якщо раніше тестування розглядалося як допоміжний процес, то сьогодні воно перетворюється на критично важливий елемент забезпечення якості програмного забезпечення. Це пов'язано з тенденціями мікросервісної архітектури, розподілених систем та постійної інтеграції компонентів, що радикально ускладнює процеси верифікації програмних продуктів.

Методологія автоматизованого тестування еволюціонує від реактивної моделі до проактивної стратегії. Це означає принципову зміну парадигми: замість послідовного виправлення помилок, тестування стає вбудованим процесом, який супроводжує розробку на всіх її етапах. Ключову роль у цій трансформації відіграють підходи безперервної інтеграції та доставки (CI/CD), які дозволяють автоматизувати процеси верифікації якості.

1.3 Огляд існуючих підходів

Існує кілька підходів до оцінки покриття коду, зокрема покриття інструкцій, гілок, шляхів, функцій та класів. Кожен з цих підходів має свої переваги та обмеження:

- покриття інструкцій (Statement Coverage): Вимірює кількість виконаних інструкцій у програмному коді під час тестування. Цей підхід є найпростішим і часто використовується, але не забезпечує перевірку всіх можливих шляхів виконання програми;
- покриття гілок (Branch Coverage): Вимірює кількість виконаних гілок умовних операторів. Цей підхід дозволяє перевірити логічні умови програми, але не завжди охоплює всі можливі комбінації умов;
- покриття шляхів (Path Coverage): Вимірює кількість виконаних шляхів через програму. Цей підхід є найбільш повним, але водночас найскладнішим для реалізації, оскільки кількість можливих шляхів може бути дуже великою;

- покриття функцій (Function Coverage): Вимірює кількість виконаних функцій у програмі. Цей підхід забезпечує загальний огляд тестування, але може пропустити детальні перевірки всередині функцій;
- покриття класів (Class Coverage): Вимірює кількість виконаних класів у програмі. Цей підхід корисний для об'єктно-орієнтованого програмування, але не забезпечує детальне покриття методів всередині класів.

1.4 Обмеження існуючих підходів

Незважаючи на значимість методів оцінки покриття коду, сучасні підходи мають низку суттєвих обмежень, що ускладнюють забезпечення абсолютної надійності програмних систем:

- неповнота покриття логічних сценаріїв: Жоден з наявних підходів не може гарантувати стовідсоткове покриття всіх можливих логічних шляхів виконання програми;
- економічна неефективність складних методик: Деякі методології покриття коду, зокрема аналіз всіх можливих шляхів виконання, мають значні обмеження;
- критична залежність від якості тестових сценаріїв: Формальні показники покриття коду не є гарантією реальної якості та ефективності тестування;
- технологічні обмеження інструментарію: Наявні засоби оцінки покриття коду мають суттєві обмеження при роботі.

Подолання цих обмежень вимагає комплексного, міждисциплінарного підходу, що поєднує технологічні інновації, методологічні вдосконалення та врахування специфіки конкретних проєктних контекстів.

1.5 Тенденції та перспективи

Серед ключових тенденцій, що визначають напрямки еволюції галузі, можна виокремити наступні:

Глибока інтеграція з CI/CD процесами: Сучасні підприємства все активніше впроваджують комплексні методології безперервної інтеграції та доставки, де оцінка покриття коду стає невід'ємним елементом загального конвеєра розробки. Це дозволяє забезпечити не лише постійний моніторинг якості програмного забезпечення, але й створити систему випереджувального реагування на потенційні дефекти та ризики.

Еволюція інструментарію оцінки покриття коду: Спостерігається стрімкий розвиток спеціалізованих інструментальних засобів, таких як Codeception Code coverage для PHP, Istanbul для JavaScript та інтегровані рішення для різних мов програмування. Сучасні інструменти демонструють тенденцію до:

- підвищення точності вимірювання покриття коду;
- розширення функціональних можливостей аналізу;
- спрощення інтеграції в існуючі екосистеми розробки;
- забезпечення деталізованої та наочної звітності.

Впровадження технологій штучного інтелекту та машинного навчання: Застосування інтелектуальних алгоритмів відкриває принципово нові можливості в аналізі покриття коду, зокрема:

- прогнозування потенційних дефектів на ранніх стадіях розробки;
- автоматизована генерація тестових сценаріїв;
- інтелектуальний аналіз складних залежностей в архітектурі додатків;
- оптимізація тестових стратегій на основі машинного навчання.

Перспективи розвитку галузі пов'язані з подальшою автоматизацією, інтелектуалізацією та інтеграцією процесів тестування в життєвий цикл розробки програмного забезпечення. Очікується, що найближчим часом домінуючими трендами стануть контекстно-орієнтовані підходи до оцінки покриття коду, які враховуватимуть специфіку конкретних проєктів та технологічних стеків.

1.6 Масштаб проблеми

Проблема забезпечення високого рівня покриття коду в контексті автоматизованого тестування веб-застосунків є надзвичайно актуальною та

комплексною науково-практичною проблемою сучасної інженерії програмного забезпечення. Її актуальність зумовлена перманентним ускладненням архітектури веб-додатків, зростанням вимог до якості та надійності програмних продуктів, а також необхідністю оптимізації процесів розробки та тестування [3].

Масштабність досліджуваної проблеми виявляється в багатовимірності її аспектів, що охоплюють технічні, методологічні та економічні площини. Вона включає глибокий аналіз не лише технічних механізмів вимірювання покриття коду, але й організаційних принципів інтеграції практик тестування в життєвий цикл розробки програмного забезпечення.

Основними викликами, що ускладнюють досягнення та підтримку високого рівня покриття коду в сучасних веб-застосунках, є:

- комплексність архітектури систем: Сучасні веб-застосунки характеризуються високою складністю архітектурних рішень, що включають мікросервісні архітектури, розподілені системи, складні фреймворки та безліч інтеграційних компонентів. Це принципово ускладнює процес створення повномасштабних тестових сценаріїв, які здатні охопити всі можливі варіанти виконання коду;
- динамічність технологічного ландшафту: Постійні зміни в технологіях веб-розробки, оновлення мов програмування, фреймворків та бібліотек вимагають перманентної адаптації тестових стратегій. Тестувальники змушені регулярно переглядати та оновлювати тестові набори, що робить процес підтримки code coverage надзвичайно трудомістким і ресурсоємним;
- обмеження ресурсного забезпечення: Реалізація концепції повного покриття коду потребує значних інвестицій часу, кваліфікованих людських ресурсів та спеціалізованого інструментарію. Малі та середні компанії часто стикаються з браком бюджетних та кадрових можливостей для впровадження повномасштабних практик code coverage;

- методологічні та процесні бар'єри: Існують суттєві розбіжності в розумінні та впровадженні практик тестування між різними командами. Відсутність уніфікованих стандартів та методологій створює додаткові перешкоди для забезпечення високоякісного покриття коду;
- технологічні обмеження інструментів: Наявні інструменти вимірювання code coverage мають певні технічні обмеження, особливо при роботі з складними фреймворками, асинхронним кодом та динамічними мовами програмування. Це ускладнює точне вимірювання реального рівня покриття коду.
- дослідження масштабу проблеми code coverage в автоматизованому тестуванні веб-застосунків є критично важливим для підвищення якості та надійності програмних продуктів, оптимізації процесів розробки та мінімізації ризиків.

1.7 Рівень практичності дослідження

Дослідження ефективності code coverage в автоматизованому тестуванні веб-застосунків є актуальним науковим напрямком, що має суттєвий потенціал для вирішення практичних проблем у галузі розробки програмного забезпечення.

Практична значущість роботи визначається можливістю оптимізації процесів тестування в реальних проєктах. Дослідження надасть конкретні рекомендації щодо вибору оптимальних стратегій тестового покриття, що дозволить тестувальникам більш ефективно виявляти та попереджати потенційні дефекти на ранніх стадіях розробки.

Особливу цінність становить розроблена методика інтеграції інструментів оцінки покриття коду в процеси безперервної інтеграції та доставки (CI/CD). Це наддасть можливість автоматизувати моніторинг якості програмного забезпечення, скоротити витрати на тестування та підвищити загальну надійність веб-застосунків.

Аналіз предметної галузі показує, що оцінка покриття коду є ключовим компонентом автоматизованого тестування веб-застосунків, який дозволяє

забезпечити високу якість програмного забезпечення. Існуючі підходи та інструменти для вимірювання покриття коду мають свої переваги та обмеження, але загалом сприяють покращенню процесу розробки. Інтеграція оцінки покриття коду в процеси CI/CD для аналізу покриття є перспективними напрямками, що можуть значно підвищити ефективність тестування і забезпечити високу якість веб-застосунків.

Застосування результатів дослідження можливе не лише у великих IT-компаніях, але й у невеликих командах розробки, які прагнуть підвищити якість свого продукту без значного збільшення витрат. Методики, описані у роботі, є масштабованими та можуть бути адаптовані під різні технологічні стекі та етапи життєвого циклу розробки. Це робить результати дослідження універсальними і корисними для широкого кола фахівців, включаючи тестувальників, DevOps-інженерів та розробників, які зацікавлені в удосконаленні процесів забезпечення якості.

2 ОГЛЯД Й АНАЛІЗ ЛІТЕРАТУРНИХ, НАУКОВИХ ДЖЕРЕЛ

2.1 Вплив високого покриття коду на якість веб-застосунку

Досліджуючи вплив високого охоплення коду на якість програмних продуктів, кілька джерел забезпечують повне розуміння переваг, проблем і найкращих практик, пов’язаних із охопленням коду. Ці джерела були обрані з огляду на їх авторитет у галузі, відповідність темі дослідження, об’єктивність та достовірність інформації, яку вони подають.

Авторами вибраних джерел є визнані експерти та організації з розробки та тестування програмного забезпечення. Наприклад, включено статті від Brainhub і Codecov через їхній великий досвід і внесок у цю сферу. Ці джерела дуже актуальні, оскільки вони безпосередньо стосуються ефективності та наслідків покриття коду в автоматизованому тестуванні.

Джерела, вибрані для цього огляду, надають збалансовані погляди, обговорюючи як переваги, так і обмеження високого охоплення коду. Вони вважаються надійними, оскільки базуються на емпіричних даних, практичному досвіді та загальноприйнятих галузевих практиках.

Отже, джерела можна згрупувати наступним чином.

Користь використання метрик покриття коду.

«Demystifying the Pursuit of 100% Test Code Coverage». Стаття присвячена актуальній проблемі тестування програмного забезпечення та розкриває професійний погляд досвідченого інженера на методології розробки та тестування.

Автор розпочинає з принципової тези, що досягнення 100% покриття тестами не є самостійною метою, а виступає закономірним наслідком правильної методології розробки [4]. Ключовий акцент зроблено на методології Behaviour-Driven Development (BDD), яка передбачає створення тестів на основі користувацьких сценаріїв та запроваджує якісно новий підхід до тестування.

У статті ґрунтовно проаналізовано критерії якості тестів. Зокрема, виділено три основні виміри: чутливість (здатність виявляти помилки), трасованість (можливість швидкої ідентифікації проблемних ділянок) та стійкість під час

рефакторингу. Автор наголошує, що традиційна піраміда тестування є недосконалою, і пропонує власну модель у вигляді "перевернутої груші".

Принципово важливим є висновок про necessity збереження повного покриття тестами. На думку автора, навіть мінімальне зниження покриття (на 2%) може приховувати критичні вади в логіці додатку. Метафорично цей принцип порівнюється з коротkou ковдрою, де завжди щось залишається незахищеним.

Особливу увагу приділено методології BDD, яка природним чином забезпечує максимальне покриття тестами [5]. Автор підкреслює, що тести мають безпосередньо відображати критерії прийняття користувацьких історій і створюватися випереджально.

Водночас стаття містить важливе застереження: повне покриття тестами не є абсолютною гарантією відсутності помилок. Це радше індикатор дисципліни розробки та суворого дотримання методологічних принципів. Додатковим інструментом забезпечення якості автор вважає peer-review, де колегиретельно перевіряють відповідність тестів специфікаціям.

Підсумовуючи, стаття презентує збалансований, практико-орієнтований погляд на тестування ПЗ. Основна теза полягає в тому, що 100% покриття тестами - це не формальний показник, а відображення глибокого розуміння процесів розробки, дотримання методологічної культури та прагнення до високої якості програмного продукту.

«Who Cares About Code Coverage and Why?». Покриття коду є важливим інструментом оцінки якості програмного забезпечення, який допомагає вимірювати ефективність тестування. Метрика показує частку коду, яка виконується під час тестових запусків, що дозволяє команді розробників більш точно розуміти рівень перевірки своєї системи.

Стаття розглядає значення покриття коду для різних професійних ролей у розробці програмного забезпечення. Для інженерів-розробників це насамперед означає проактивний аналіз коду, ретельну перевірку складних алгоритмічних структур, таких як розгалуження та виняткові сценарії [6]. Інженери з тестування,

своєю чергою, прагнуть максимізувати покриття, оцінюючи повноту тестування та пояснюючи колегам переваги цього підходу.

Керівники інженерних підрозділів використовують покриття коду як індикатор ефективності тестування та інструмент формування культури якості. Вони забезпечують виділення необхідних ресурсів для thorough тестування та слідкують за загальним рівнем якості програмного забезпечення.

Власники продукту розглядають покриття коду як спосіб контролю технічного боргу та балансування між розвитком нових функцій і підтриманням високої якості системи. Для них це також інструмент мотивації команди та оцінки прогресу проєкту.

Важливо розуміти, що покриття коду - це не просто формальна метрика, а комплексний підхід до забезпечення надійності та безпеки програмного забезпечення. Хоча досягнення 100% покриття в великих проєктах може бути складним, постійне прагнення до вдосконалення та особлива увага до критичних частин коду є ключовим моментом.

Стаття підкреслює колективну природу покриття коду, демонструючи, що це інструмент, який об'єднує різних фахівців заради створення більш надійного, безпечного та якісного програмного забезпечення.

«What Is Code Coverage and How to Optimize It?». За допомогою показників охоплення коду команди можуть визначити області кодової бази, які потребують додаткового тестування, і виміряти ефективність своїх зусиль з тестування. Роблячи це, вони можуть покращити якість свого програмного забезпечення, що призведе до підвищення задоволеності клієнтів, зниження витрат на обслуговування та швидшого виходу на ринок. Ви можете визначити сфери, які потрібно вдосконалити, відстежувати прогрес і переконатися, що ваше програмне забезпечення відповідає потребам ваших користувачів [7].

«On Code Coverage in Software Testing». Покриття коду є важливою метрикою, що дозволяє оцінити якість тестування програмного забезпечення. Воно допомагає визначити ділянки коду, які залишилися неперевіреними, що потенційно знижує надійність продукту.

Підвищення покриття сприяє поліпшенню якості коду, полегшує його обслуговування та знижує ризики помилок перед релізом. Регулярне використання звітів про покриття дозволяє виявляти прогалини в тестах та вдосконалювати їх [8].

Оптимальним вважається рівень покриття в межах 80–90%, хоча багато факторів впливають на реалістичність досягнення цієї мети. Повне покриття (100%) рідко доцільне, оскільки потребує значних ресурсів і не завжди виправдане в умовах комерційної розробки.

У практиці розробки рекомендовано автоматизувати збір і перевірку показників покриття у CI/CD-процесах. Це дозволяє своєчасно виявляти проблеми та контролювати якість на всіх етапах розробки.

Високе покриття саме по собі не гарантує відсутності помилок, тому його слід розглядати як один з елементів загальної стратегії забезпечення якості, разом із рев'ю коду, аналізом складності та тестуванням у продакшені.

Недоліки використання метрик покриття коду.

«Test Coverage vs Code Coverage: Everything You Need to Know». Покриття коду часто помилково ототожнюють з якістю коду. Високе покриття коду вказує на те, що під час тестування було виконано велику частину кодової бази, але це не гарантує, що тести є значущими або що код вільний від дефектів [9]. Відповідно до статті, покриття коду може призвести до хибного відчуття безпеки, оскільки воно не відображає ретельності чи ефективності проведених тестів.

«Disadvantages of Code Coverage as a Measurement». Однією з важливих проблем із охопленням коду є те, що воно може стимулювати розробників або тестувальників писати поверхневі тести, спрямовані просто на підвищення відсотка охоплення, а не на виявлення помилок. Така практика може призвести до тестів, які виконують код без перевірки його поведінки чи виявлення граничних випадків [10]. Стаття «Disadvantages of Code Coverage as a Measurement» підкреслює, що цей підхід може призвести до зосередження на досягненні високого відсотка покриття коду, а не на якості тестів.

2.2 Висновки з огляду джерел

Дослідження показують, що високий відсоток покриття коду може значно підвищити якість веб-застосунку, але цей показник не є абсолютним індикатором бездоганності програмного продукту. Таке покриття забезпечує виявлення критичних помилок і сприяє створенню стійкої та надійної кодової бази. Водночас, автори наголошують на важливості не лише кількісного, але й якісного підходу до тестування, включаючи чутливість тестів, їхню трасованість і стійкість до змін у коді.

Також важливо дослідити взаємозв'язок між високим покриттям коду та реальним зниженням технічного боргу і кількістю помилок у продуктивному середовищі. Це дозволить визначити, які саме метрики покриття коду є найефективнішими для забезпечення якості програмного забезпечення.

Отже, подальші дослідження повинні зосередитися на вивченні якісних аспектів тестування, таких як ефективність тестів, їхня здатність виявляти критичні помилки, а також вплив різних методологій розробки на покриття коду. Це допоможе забезпечити більш комплексний і надійний підхід до оцінки якості програмного забезпечення.

3 ПОСТАНОВКА ЗАДАЧІ

3.1 Оцінка та порівняння інструментів для вимірювання покриття коду

Я обрав мову програмування PHP для дослідження ефективності покриття коду в автоматизованому тестуванні веб-застосунків, оскільки PHP є однією з найпопулярніших мов для веб-розробки, широко використовується у багатьох сучасних веб-застосунках і системах управління контентом. Крім того, PHP має добре розвинену екосистему інструментів для тестування та вимірювання покриття коду, що забезпечує точність, інтеграцію з CI/CD пайплайнами, можливості візуалізації та сумісність з Docker, що є важливим для сучасних розробницьких середовищ. Зокрема, я обрав середовище Codeception, яке є потужним фреймворком для тестування, що підтримує різні типи тестів (функціональні, інтеграційні, приймальні) та забезпечує зручний інтерфейс для роботи з тестовими сценаріями, дозволяючи ефективно вимірювати та аналізувати покриття коду.

При виборі інструменту для оцінки покриття коду було розглянуто кілька популярних альтернатив, серед яких:

- Xdebug: Популярний PHP-дебаггер, який також використовується для вимірювання покриття коду. Забезпечує високу точність, але значно впливає на час виконання тестів;
- PHPUnit Coverage: Інструмент для аналізу покриття, який добре інтегрується з Codeception. Має стабільну точність, але обмежені можливості візуалізації;
- Codecov: Платформа для управління звітами покриття, що забезпечує хмарне зберігання та аналіз результатів тестів. Підтримує інтеграцію з CI/CD і надає розширені можливості візуалізації;
- PHP Debugger: Інструмент для налагодження PHP-коду з можливостями вимірювання покриття, але впливає на продуктивність;
- PCOV: Легкий інструмент для вимірювання покриття, швидший за Xdebug, але має обмежені можливості для налагодження.

Для оцінки альтернатив розглядалися такі критерії:

- точність вимірювання покриття коду. Це головний критерій, оскільки точне визначення покриття коду важливе для оцінки якості тестування. Інструмент повинен відображати всі можливі шляхи виконання та функції, які не покриті тестами, щоб забезпечити максимально повну перевірку;
- вплив на час виконання тестів. Цей критерій відображає наскільки використання інструменту для вимірювання покриття коду впливає на загальний час виконання тестів. Вимірюється як відсоткове збільшення часу виконання порівняно з базовим часом (без вимірювання покриття);
- можливість інтеграції з CI/CD пайплайнами. Інтеграція з CI/CD системами, такими як Jenkins або GitLab CI, є важливою для автоматизації процесів тестування і дозволяє постійно відстежувати покриття коду у проєкті, забезпечуючи контроль якості на кожному етапі розробки;
- можливості візуалізації та аналітики. Інструмент повинен надавати зручний інтерфейс або інтеграцію для візуалізації результатів, що полегшує аналіз даних про покриття коду. Графічне відображення, інтерактивні звіти та історичні дані допомагають у визначенні прогалин у покритті та прийнятті рішень щодо подальшого тестування;
- сумісність із Docker. Використання Docker в проєктах стає стандартом для забезпечення однакового середовища розробки та тестування. Інструмент для покриття коду має підтримувати роботу у Docker-контейнерах, щоб забезпечити стабільну та передбачувану поведінку тестів незалежно від середовища.

Шкала критеріїв представлена в таблиці 3.1

Таблиця 3.1 – Шкали критеріїв

Шкала критерія	Тип шкали	Значення
Точність вимірювання покриття коду	Шкала відносин	90% і більше – високий рівень точності покриття, що означає мінімальні прогалини.
		70%-89% – середній рівень точності, який вказує на можливі прогалини в покритті.
		Менше 70% – низький рівень точності, що може свідчити про значні прогалини в тестуванні.
Вплив на час виконання тестів	Шкала відносин	Збільшення до 25% – мінімальний вплив на продуктивність, оптимально для CI/CD процесів.
		Збільшення від 25% до 75% – помірний вплив на швидкість, прийнятний для більшості проєктів.
		Збільшення більше 75% – значний вплив на продуктивність, може створювати затримки в процесі розробки.
Можливість інтеграції з CI/CD пайплайнами	Номінальна шкала	Легка інтеграція = 10
		Часткова інтеграція = 5
		Відсутня інтеграція = 1
Можливості візуалізації та аналітики	Порядкова шкала	Високий рівень деталізації = 10
		Середній рівень деталізації = 5
		Низький рівень деталізації = 1
Сумісність із Docker	Номінальна шкала	Повна сумісність = 10
		Часткова сумісність = 5
		Відсутня сумісність = 1

Отже в результаті векторного опису альтернатив, маємо такі дані, що представлені в таблиці 3.2

Таблиця 3.2 – Векторний опис альтернатив

Альтернатива	Точність вимірювання покриття коду(%)	Вплив на час виконання тестів(%)	Можливість інтеграції з CI/CD	Можливості візуалізації та аналітики	Сумісність із Docker
Xdebug	85	60	5	5	5
PHPUnit Coverage	90	30	10	5	10
Codecov	88	20	10	10	10
PHP Debugger	80	50	10	5	10
PCOV	70	10	1	1	5

За принципом Парето можна зменшити кількість варіантів, ми можемо викреслити PCOV. PVOC значно поступається іншим інструментам за точністю, інтеграцією з CI/CD, візуалізацією та сумісністю з Docker. Тому доцільно прибрати PVOC на користь більш ефективних інструментів, таких як PHPUnit Coverage, Codecov або Xdebug, які демонструють кращі результати у всіх критеріях (див.табл.3.3).

Для постановки задачі ми можемо використати метод пропорційних вагових коефіцієнтів, спочатку нормалізуємо дані, а потім обчислимо вагові коефіцієнти і використаємо їх для розрахунку корисності кожної альтернативи.

Максимальні значення для кожного критерію:

- точність вимірювання покриття коду: 90;
- вплив на час виконання тестів: 60;

- можливість інтеграції з CI/CD: 10;
- можливості візуалізації та аналітики: 10;
- сумісність із Docker: 10.

Таблиця 3.3 – Результати перевірки Парето

Альтернатива	Точність вимірювання покриття коду(%)	Вплив на час виконання тестів(%)	Можливість інтеграції з CI/CD	Можливості візуалізації та аналітики	Сумісність із Docker
Xdebug	85	60	5	5	5
PHPUnit Coverage	90	30	10	5	10
Codecov	88	20	10	10	10
PHP Debugger	80	50	10	5	10

Нормалізуємо кожне значення, ділячи на максимальне значення відповідного критерію (див.табл.3.4).

Таблиця 3.4 – Нормалізовані значення

Альтернатива	Точність вимірювання покриття коду(%)	Вплив на час виконання тестів(%)	Можливість інтеграції з CI/CD	Можливості візуалізації та аналітики	Сумісність із Docker
Xdebug	0.944	1	0.5	0.5	0.5
PHPUnit Coverage	1	0.5	1	0.5	1
Codecov	0.978	0.333	1	1	1
PHP Debugger	0.889	0.833	1	0.5	1

Припустимо, що вагові коефіцієнти для кожного критерію однакові, тобто кожен критерій має вагу w_i=1/n, де n – кількість критеріїв. В даному випадку n=5 тому w_i=1/5=0.2.

Формула 3.1 представляє обчислення корисності альтернативи z_j шляхом взваженого сумування показників альтернативи

$$z_j = \sum_{i=1}^{n} w_i a_{ij} \tag{3.1}$$

де z_j – загальна корисність альтернативи j.

n – кількість критеріїв оцінки.

w_i – вага критерію i, що відображає важливість цього критерію у загальній оцінці.

a_{ij} – нормалізоване значення критерію i для альтернативи j.

Результати розрахунку корисності для кожної альтернативи представлено в таблиці 3.5

Таблиця 3.5 – Результати розрахунку корисності для кожної альтернативи

Альтернатива	Корисність
Xdebug	(0.944 * 0.2) + (1.0 * 0.2) + (0.5 * 0.2) + (0.5 * 0.2) + (0.5 * 0.2) = 0.888
PHPUnit Coverage	(1.0 * 0.2) + (0.5 * 0.2) + (1.0 * 0.2) + (0.5 * 0.2) + (1.0 * 0.2) = 0.8
Codecov	(0.978 * 0.2) + (0.333 * 0.2) + (1.0 * 0.2) + (1.0 * 0.2) + (1.0 * 0.2) = 0.8626
PHP Debugger	(0.889 * 0.2) + (0.833 * 0.2) + (1.0 * 0.2) + (0.5 * 0.2) + (1.0 * 0.2) = 0.8444

Отже, по результатам розрахунків, найкращим інструментом для аналізу коду буде Xdebug. Цей вибір базується на його високих показниках точності

вимірювання покриття коду, а також збалансованому впливу на час виконання тестів. Незважаючи на нижчі оцінки в категоріях інтеграції з CI/CD та візуалізації, Xdebug виявився оптимальним варіантом завдяки своїй сумісності з Docker і загальним ваговим коефіцієнтам, що перевершують інші інструменти в даному контексті.

3.2 Постановка задачі

Після аналізу інструментів можна поставити задачі, які будуть виконанні в процесі роботи. Метою цієї роботи є визначення зв'язку між рівнем покриття коду та якістю веб-застосунків, а також виявлення основних проблем, пов'язаних з досягненням і підтримкою високого покриття коду. У процесі дослідження буде використовуватись мова програмування PHP, фреймворк Codeception для автоматизованого тестування, а також інструмент Xdebug для оцінки покриття коду.

Отже, для проведення дослідження необхідно вирішити наступні задачі:

- проаналізувати залежність між рівнем покриття коду та якістю веб-застосунків;
- аналіз метрик з реального проекту;
- дослідження проблем, пов'язаних з досягненням високого покриття;
- обрати тестовий веб-застосунок на мові PHP, впровадити automated tests з використанням фреймворку Codeception, використати інструмент Xdebug для вимірювання та аналізу покриття коду, провести серію експериментів з різними рівнями code coverage;
- розробка рекомендацій щодо інтеграції покриття коду в процеси CI/CD для забезпечення постійного моніторингу якості в середовищі автоматизованого тестування.

4 ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ

Для проведення дослідження ефективності покриття коду в автоматизованому тестуванні веб-застосунків було обрано мову програмування PHP, фреймворк Codeception та інструмент Xdebug для вимірювання покриття. В якості тестового веб-застосунку було обрано open-source проєкт Article Hub. А також взяті метрики з реального проекту для аналізу.

4.1 Аналізу метрик реального проекту

Було проведено аналіз метрик з реального проєкту до та після впровадження механізмів збору та контролю покриття коду (code coverage).
Для цього порівнювались три ключові показники:

– середня кількість багів на місяць;
– відсоток багів від загальної кількості задач;
– рівень покриття коду автоматизованими тестами (Coverage).

На графіку (див.рис.4.1) зображено зміну кожної з трьох метрик.

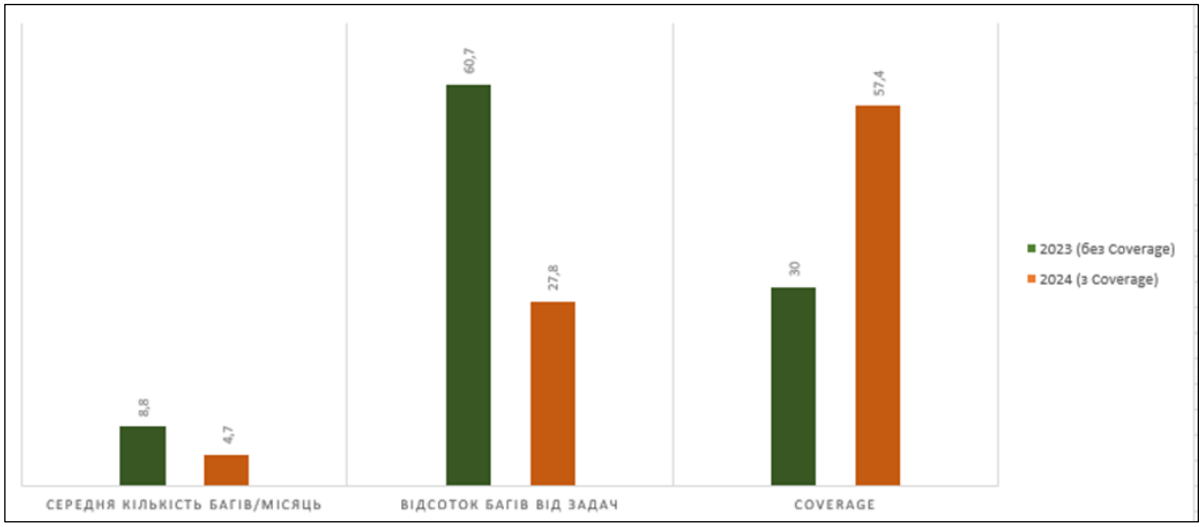


Рисунок 4.1 – Графік метрик

З наведеного графіку видно, що після впровадження code coverage середня кількість багів на місяць знизилась з 8,8 до 4,7, що становить скорочення приблизно на 47,2%. Відсоток багів від задач зменшився з 60,7% до 27,8%

(зниження на 54,2%). Водночас рівень code coverage зріс з 30% до 57,4% – приріст склав 91,4%.

Стовпчиковий графік дозволяє наочно порівняти абсолютні значення показників до і після впровадження coverage. Особливо помітною є тенденція до зменшення кількості помилок та одночасного зростання покриття коду.

Для більш глибокого аналізу було також побудовано графік динаміки змін показників протягом 2023–2024 років. Це дозволяє простежити розвиток процесу у часі та виявити ключові моменти впливу coverage (див.рис. 4.2, 4.3).

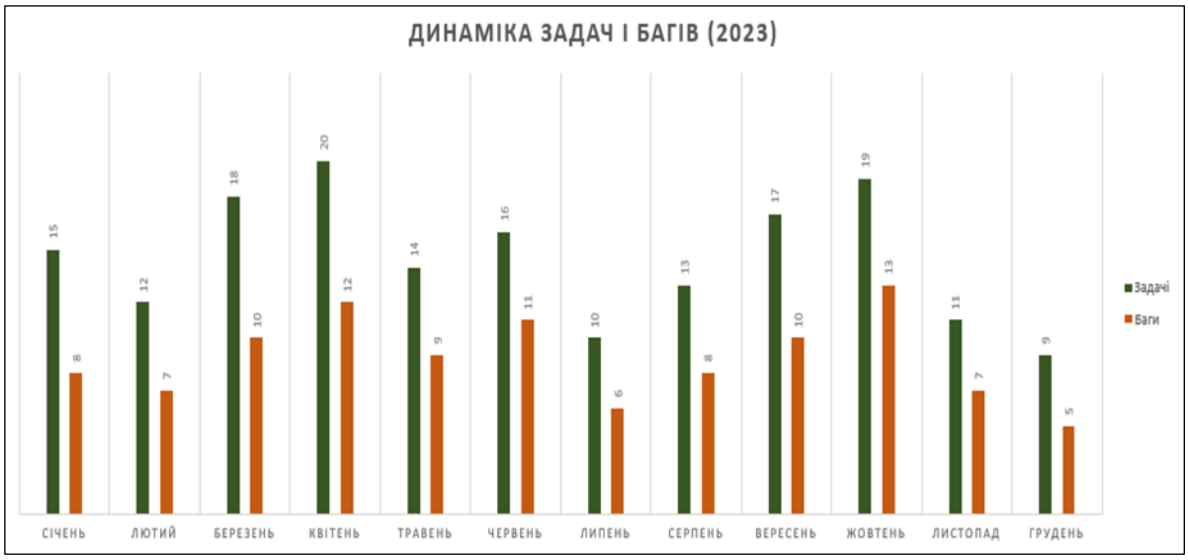


Рисунок 4.2 – Динаміка задач і багів 2023 рік

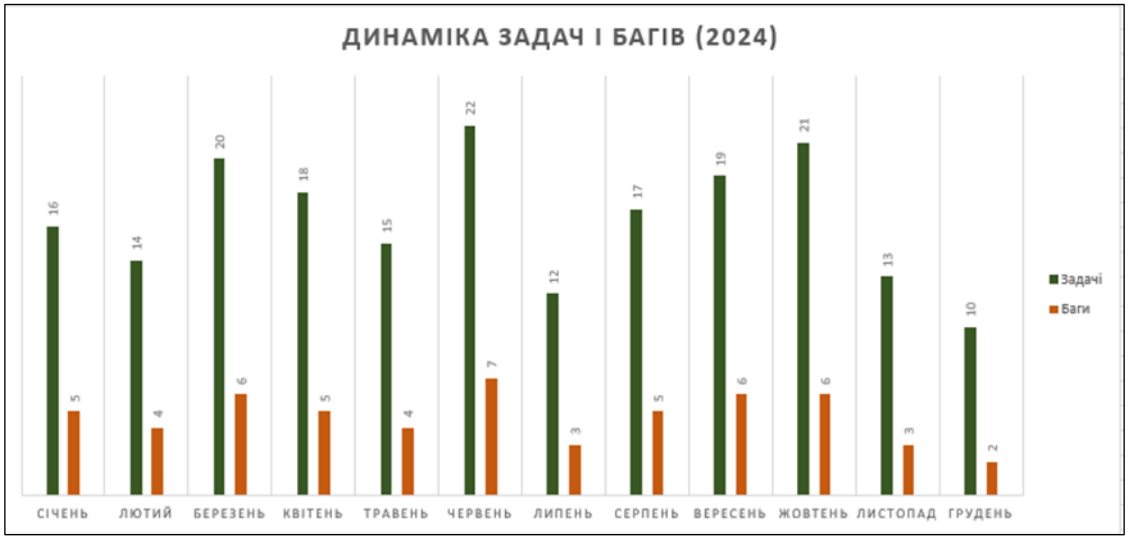


Рисунок 4.3 – Динаміка задач і багів 2024 рік

Як видно з графіків, після впровадження code coverage спостерігається стабільне зниження кількості багів. Найбільш відчутний спад стався у другому кварталі 2024 року, коли coverage перевищив 50%. Водночас покриття коду демонструє поступове зростання, що підтверджує ефективність автоматизованих підходів до тестування.

Загалом, отримані дані підтверджують гіпотезу про те, що використання code coverage у реальних умовах проєкту призводить до зменшення кількості дефектів, підвищення прозорості контролю якості та стабілізації тестового середовища.

Разом з тим, варто зазначити, що coverage є лише індикатором і не гарантує автоматичну якість. Важливим є наповнення тестів змістом, охоплення граничних сценаріїв та підтримка тестової бази в актуальному стані.

4.2 Опис обраних методів, технологій, підходів, рішень та алгоритмів

PHP (Hypertext Preprocessor) є однією з найпоширеніших мов програмування для розробки веб-застосунків. Основні переваги PHP включають простоту використання, широкий спектр бібліотек та фреймворків, активну спільноту та підтримку з боку хостинг-провайдерів. PHP дозволяє швидко створювати динамічні веб-сторінки та серверні частини веб-застосунків

Codeception – це фреймворк для автоматизованого тестування, який підтримує різні типи тестування, зокрема юніт-тестування, функціональне тестування та приймальне тестування. Codeception базується на PHPUnit та інтегрує його можливості з додатковими інструментами для спрощення написання тестів.

Xdebug – це розширення для PHP, яке надає можливості для налагодження та профілювання коду, а також вимірювання покриття коду.

Xdebug генерує детальні звіти про покриття коду, що дозволяє розробникам і тестувальникам бачити, які частини коду були виконані під час тестування. Це допомагає виявити незатестовані ділянки коду та забезпечити більш повне покриття тестами. У нашому дослідженні Xdebug використовується для вимірювання покриття коду тестами, написаними за допомогою Codeception.

Article Hub – це тестовий веб-застосунок, розроблений на основі фреймворку Symfony, який дозволяє користувачам створювати, оцінювати, писати коментарі та підписуватися на статті [11]. Основні можливості Article Hub включають:

- створення статей: Користувачі можуть створювати нові статті, додаючи заголовок, вміст та інші необхідні метадані;
- оцінювання статей: Користувачі мають можливість оцінювати статті, надаючи їм позитивні або негативні відгуки;
- коментування статей: Користувачі можуть залишати коментарі під статтями, що дозволяє вести дискусії та ділитися думками щодо опублікованого матеріалу;
- підписка на статті: Користувачі можуть підписуватися на статті або авторів, отримуючи сповіщення про нові публікації чи оновлення.

4.2 Практична реалізація

4.2.1 Встановлення та налаштування веб-застосунку Article Hub

Склонуємо проєкт з репозеторія за допомогою команди:

```
git clone https://gitlab.com/doit9373550/ArticleHub.git
```

Після успішного клонування бачимо файли проєкту (див.рис.4.4).

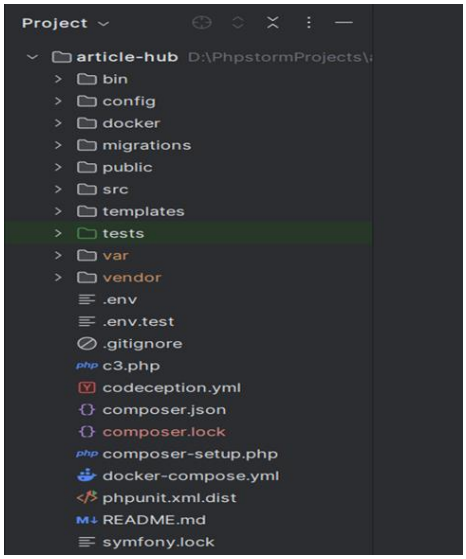


Рисунок 4.4 – Файли проєкту

Локально застосунок можна запускати за допомогою Docker:

```
docker-compose up -d
```

При переході по посиланню *http://localhost:8000/micro-post* бачимо працюючий веб-застосунок (див.рис.4.5).

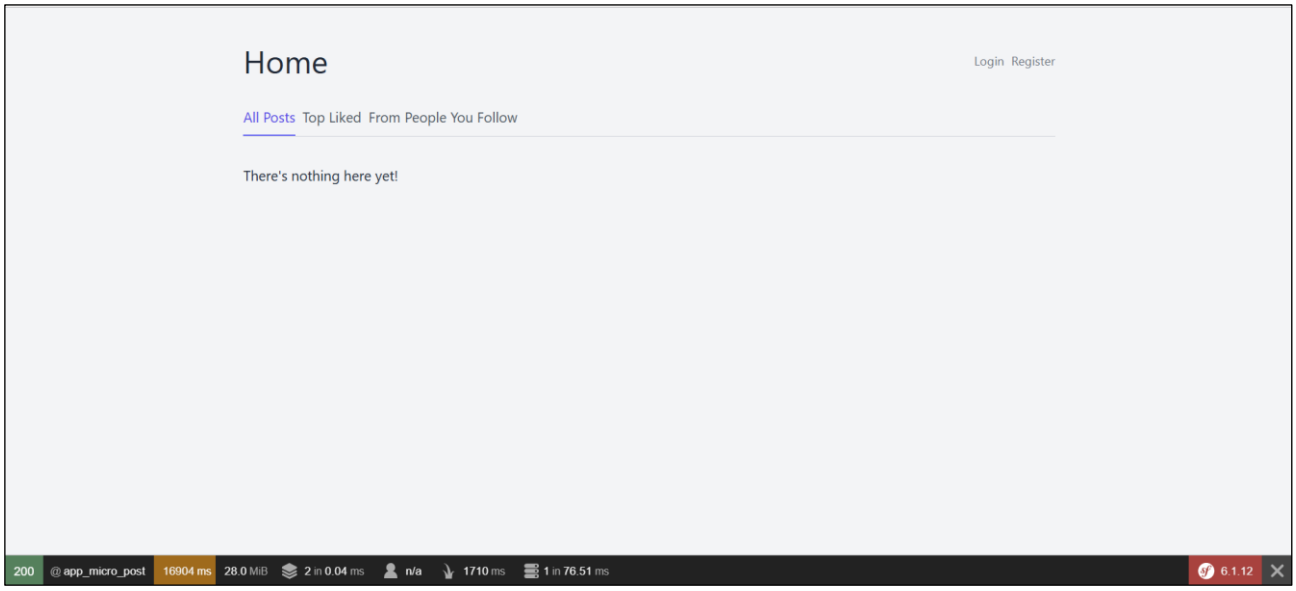


Рисунок 4.5 – Головна сторінка веб-застосунку Article Hub

4.2.2 Встановлення Codeception

Після успішного встановлення проєкту, можна встановити фреймворк Codeception, на якому будуть написані авто тести для веб-застосунку. Всю документацію можна взяти з офіційного сайту Codeception [12].

Отже почнемо встановлення фреймворку:

```
composer require "codeception/codeception" --dev
```

Наступним буде генерація файлу для налаштування Unit тестів:

```
php vendor/bin/codecept generate:suite Unit
```

Отже маємо директорію tests де і будь наші тести та Налаштування (див.рис.4.6).

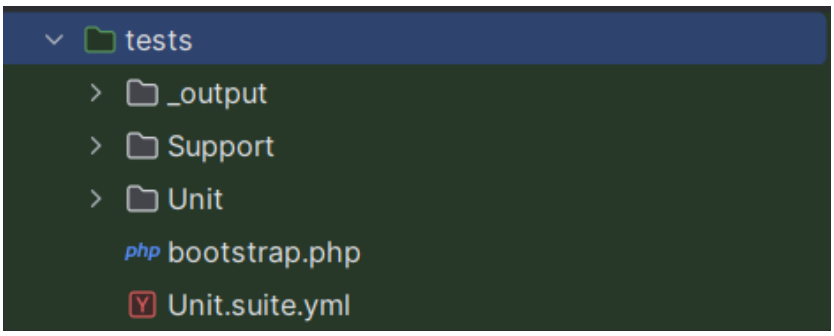


Рисунок 4.6 – Директорія tests

4.2.3 Інтеграція Xdebug та Code coverage

Оскільки я використовував Docker для локального розгортання проєкту то і вже в файлі Dockerfile було реалізована інтеграція Xdebug. Де були прописані команди встановлення відладчика та копіювання налаштувань в середину контейнеру.

```
# Встановлення Xdebug
RUN pecl install xdebug \
    && docker-php-ext-enable xdebug \
# Копіюємо конфігурацію Xdebug
COPY config/xdebug.ini /usr/local/etc/php/conf.d/
```

І також прописано завчасно налаштування Xdebug в файлі xdebug.ini.

```
[xdebug]
xdebug.mode=coverage
xdebug.start_with_request=yes
xdebug.output_dir=./debug/logs
xdebug.client_host=host.docker.internal
xdebug.log_level=0
xdebug.idekey=PHPSTORM
```

Встановлювання Code coverage в Codeception також можна подивитися в офіційних джерелах https://codeception.com/docs/Codecoverage.

В файлі codeception.yml встановити налаштування:

```
coverage:
    enabled: true
    remote: true
    include:
```

```
        - src/*
    exclude:
        - ci/*
        - config/*
        - app/cache/*
        - app/logs/*
        - tests/*
    support: tests/Support
    show_only_summary: true
    log: tests/_output
    show_uncovered: true
```

В додатку Г можна побачити повне налаштування для Codeception.

Тепер можна запускати тести з вимірюванням покриття коду для цього виконаємо команду з середини контейнеру:

```
vendor/bin/codecept run Unit --coverage
```

Запуск тестів з Code coverage представлено на рисунку 4.7

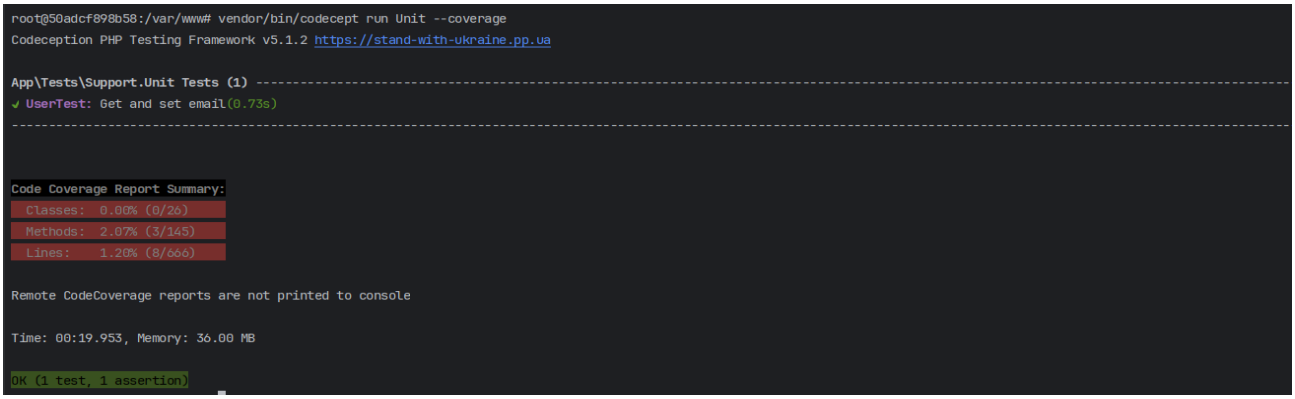


Рисунок 4.7 – Запуск тестів з Code coverage

Тут показано результати виконання юніт-тестів за допомогою фреймворку Codeception для PHP. Був запущений один тест під назвою "UserTest: Get and set email", який успішно виконався за 0.75 секунд. Підсумковий звіт про покриття коду показує дуже низьке покриття: 0% для класів, 2.07% для методів і 1.20% для рядків коду. Загальний час виконання тестів становив близько 20 секунд, використана пам'ять - 36 MB. Результат свідчить про те, що тестування пройшло успішно, але значна частина коду залишилася непокритою тестами.

4.2.4 Виконання тестів з використанням Code coverage

Напишемо більше unit тестів, наприклад для класу User.

```
class UserTest extends Unit
{
    protected UnitTester $tester;
    protected User $user;

    protected function setUp(): void
    {
        parent::setUp();
        $this->user = new User();
    }

    public function testGetAndSetEmail()
    {
        $email = "test@example.com";
        $this->user->setEmail($email);
        $this->assertSame($email, $this->user->getEmail());
    }

    public function testGetAndSetPassword()
    {
        $password = "Qwerty1!";
        $this->user->setPassword($password);
        $this->assertSame($password, $this->user->getPassword());
    }

    public function testGetAndSetIsVerified()
    {
        $this->user->setIsVerified(true);
        $this->assertTrue($this->user->isVerified());
    }

    public function testAddAndRemoveFollower()
    {
        $follower = new User();

        $this->user->addFollower($follower);
        $this->assertTrue($this->user->getFollowers()-
>contains($follower));

        $this->user->removeFollower($follower);
        $this->assertFalse($this->user->getFollowers()-
>contains($follower));
    }

    public function testFollowAndUnfollow()
    {
        $followed = new User();

        $this->user->follow($followed);
        $this->assertTrue($this->user->getFollows()->contains($followed));
```

```
        $this->user->unfollow($followed);
        $this->assertFalse($this->user->getFollows()->contains($followed));
    }
}
```

Та запустимо код:

```
vendor/bin/codecept run tests/Unit --coverage-html
```

Запускав тести з прапором --coverage-html для отримання звіт з покриття коду ще і як HTML сторінка для більшого розуміння, які частини коду покритті а які ні. Результати які отримали на виході (див.рис.4.8):

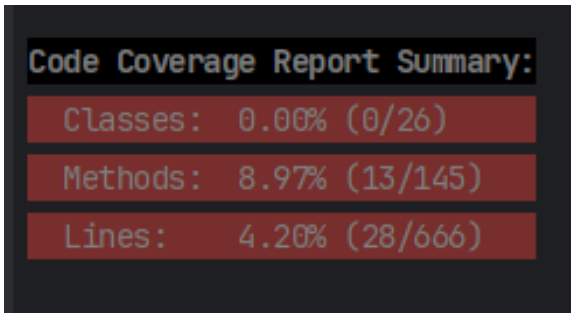


Рисунок 4.8 – Звіт з Code coverage після написання тестів

І в директорії tests/_output/coverage автоматично генерується HTML звіт, який можна продивитися запустивши index.html (див.рис.4.9)

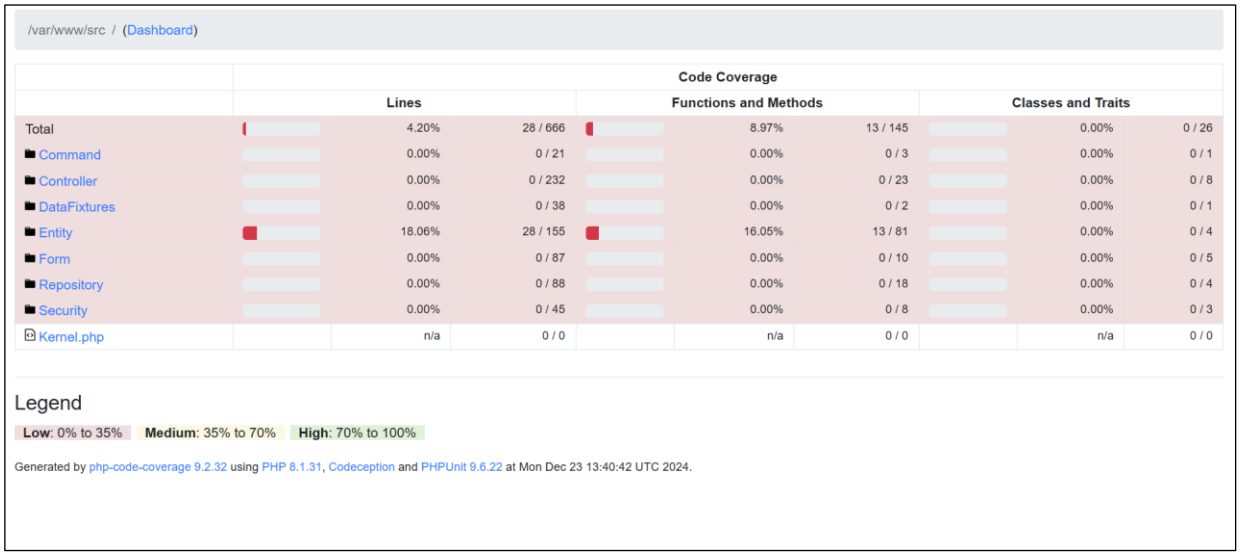


Рисунок 4.9 – HTML сторінка згенерована Code coverage

HTML звіт, згенерований інструментом Code coverage, є потужним засобом для візуалізації покриття тестами вашого коду. Він надає детальну інформацію про те, які частини вашого проєкту покриті тестами, а які залишаються неперевіреними. Цей звіт дозволяє розробникам легко ідентифікувати слабкі місця у своїх тестах та прийняти обґрунтовані рішення щодо подальшого вдосконалення тестового покриття.

На головній сторінці звіту можна побачити загальну статистику покриття коду для всього проєкту. Тут відображаються такі показники, як відсоток покриття коду тестами, кількість покритих та непокритих файлів, класів, методів та рядків коду. Ця інформація дає загальне уявлення про стан тестового покриття у проєкті.

Звіт також надає можливість перегляду покриття коду по окремих директоріях проєкту. Це дозволяє детально аналізувати, які частини проєкту мають недостатнє покриття. Для кожної директорії відображається статистика покриття, що дозволяє швидко визначити, де необхідно зосередити зусилля для написання додаткових тестів (див.рис.4.10).

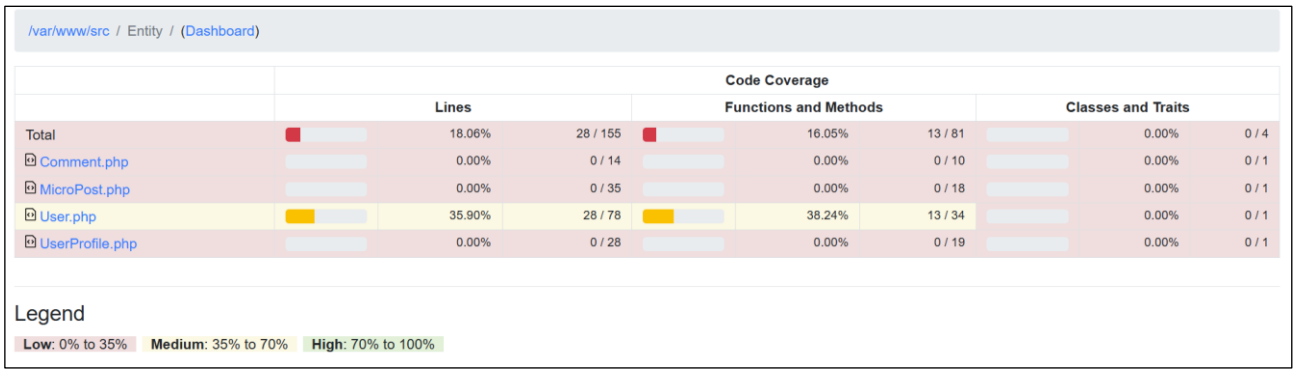
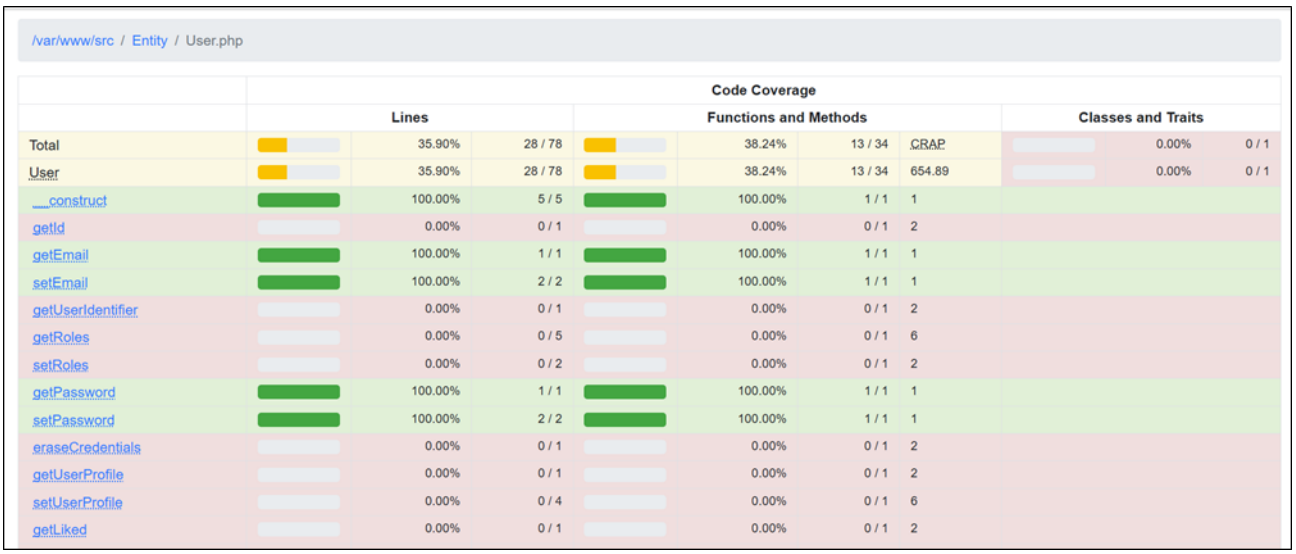


Рисунок 4.10 – Статистика покриття окремої директорії

HTML звіт дозволяє детально переглянути покриття для кожного файлу у проєкті. Відкривши конкретний файл, ви можете побачити, які рядки коду були виконані під час тестування, а які залишилися непокритими. Покриті тестами рядки зазвичай виділяються зеленим кольором, а непокриті – червоним. Це допомагає швидко ідентифікувати місця, які потребують додаткових тестів (див.рис.4.11).



/var/www/src / Entity / User.php

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total	35.90%	28 / 78		38.24%	13 / 34	CRAP	0.00%	0 / 1	
User	35.90%	28 / 78		38.24%	13 / 34	654.89	0.00%	0 / 1	
__construct	100.00%	5 / 5		100.00%	1 / 1	1			
getId	0.00%	0 / 1		0.00%	0 / 1	2			
getEmail	100.00%	1 / 1		100.00%	1 / 1	1			
setEmail	100.00%	2 / 2		100.00%	1 / 1	1			
getUserIdentifier	0.00%	0 / 1		0.00%	0 / 1	2			
getRoles	0.00%	0 / 5		0.00%	0 / 1	6			
setRoles	0.00%	0 / 2		0.00%	0 / 1	2			
getPassword	100.00%	1 / 1		100.00%	1 / 1	1			
setPassword	100.00%	2 / 2		100.00%	1 / 1	1			
eraseCredentials	0.00%	0 / 1		0.00%	0 / 1	2			
getUserProfile	0.00%	0 / 1		0.00%	0 / 1	2			
setUserProfile	0.00%	0 / 4		0.00%	0 / 1	6			
getLiked	0.00%	0 / 1		0.00%	0 / 1	2			

Рисунок 4.11 – Статистика покриття окремного файлу

Звіт також надає можливість перегляду покриття для окремих класів та методів. Це особливо корисно для великих проєктів, де необхідно детально аналізувати тестове покриття на рівні окремих компонентів. Перегляд покриття на рівні методів дозволяє зрозуміти, які логічні частини коду залишаються неперевіреними.

Аналізуючи звіт покриття коду тестами, представлений на рисунку 9, було виявлено відсутність тестового покриття для методу setRoles у класі User.php. Даний метод є критично важливим компонентом системи управління правами доступу, оскільки відповідає за коректне призначення ролей користувачам у системі.

Відсутність належного тестового покриття для такого критичного методу створює потенційні ризики для безпеки та стабільності роботи системи. При внесенні змін до функціоналу управління ролями існує висока ймовірність виникнення помилок, які можуть призвести до некоректного призначення прав доступу. Це, в свою чергу, може мати серйозні наслідки для системи, зокрема:

– порушення принципів розмежування доступу, що може призвести до несанкціонованого доступу до функціоналу системи;
– некоректна робота механізмів авторизації, що може вплинути на безпеку даних користувачів;

– потенційні збої в роботі пов'язаних підсистем, які залежать від коректного функціонування механізму ролей.

Тоді треба додати тест на цю функцію:

```
public function testSetRoles()
{
    // Test setting single role
    $singleRole = ['ROLE_USER'];
    $this->user->setRoles($singleRole);
    $this->assertSame($singleRole, $this->user->getRoles());

    // Test setting multiple roles
    $multipleRoles = ['ROLE_USER', 'ROLE_ADMIN', 'ROLE_WRITER'];
    $this->user->setRoles($multipleRoles);
    $this->assertSame($multipleRoles, $this->user->getRoles());

    // Test setting empty array of roles
    $emptyRoles = [];
    $this->user->setRoles($emptyRoles);
    // User should always have at least ROLE_USER
    $this->assertSame(['ROLE_USER'], $this->user->getRoles());
}
```

Після запуску тестів і генерації звіту з покриття коду, можна побачити що тепер функція setRoles покрита (див.рис.4.12):

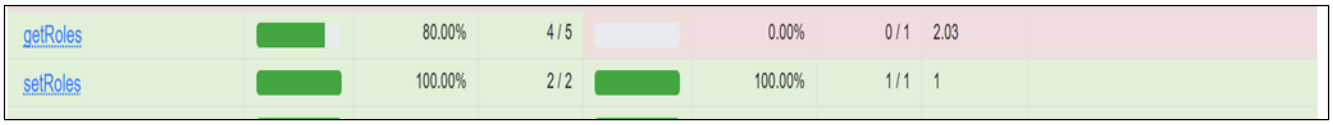


Рисунок 4.12 – Покритті тестами функції

Впровадження тестового покриття для методу setRoles у класі User.php демонструє ефективність систематичного підходу до забезпечення якості програмного забезпечення. Після додавання відповідного тестового сценарію, інструмент code coverage підтвердив повне покриття даного методу, що має кілька важливих вплив для проєкту.

По-перше, тестове покриття методу setRoles забезпечує валідацію критично важливого функціоналу управління ролями користувачів. Реалізований тестовий сценарій перевіряє різні варіанти використання методу, включаючи встановлення одиночної ролі, множинних ролей та граничний випадок з порожнім масивом

ролей. Це дозволяє гарантувати коректну роботу системи авторизації в різних ситуаціях.

По-друге, наявність автоматизованих тестів для даного методу значно підвищує надійність системи при подальшій розробці та рефакторингу. Будь-які зміни, що потенційно можуть вплинути на функціонал управління ролями, будуть одразу виявлені під час виконання тестів. Це суттєво знижує ризики випадкового порушення логіки роботи системи безпеки.

Крім того, впровадження тестового покриття для такого критичного компонента системи демонструє важливість code coverage як метрики якості програмного забезпечення. Інструмент дозволив ідентифікувати непокриту функціональність, що могла становити потенційний ризик для безпеки системи, та підтвердив успішність вжитих заходів після додавання відповідних тестів.

Таким чином, використання code coverage як інструменту контролю якості дозволяє ефективно виявляти потенційно вразливі місця в кодовій базі та систематично працювати над їх усуненням шляхом впровадження відповідного тестового покриття. Це підтверджує важливість інтеграції аналізу покриття коду в процес розробки програмного забезпечення як невід'ємної частини забезпечення його якості та надійності.

4.2.5 Рекомендації щодо метрик покриття коду

У процесі розробки програмного забезпечення метрики покриття коду відіграють ключову роль у забезпеченні якості продукту. Розглянемо типовий звіт про покриття коду, який може отримати команда розробників або тестувальників (див.рис.4.13):

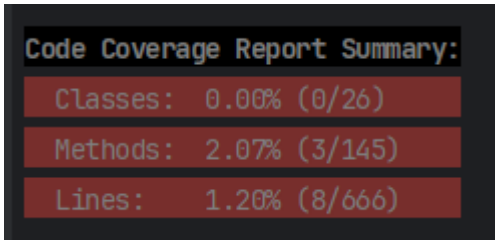


Рисунок 4.13 – Звіт з Code coverage

Даний звіт демонструє критично низькі показники покриття коду тестами, що є серйозним сигналом для команди розробки. Нульове покриття класів означає, що жоден з 26 класів системи не має повного тестового покриття. Це створює значні ризики при подальшій розробці та підтримці програмного забезпечення, оскільки будь-які зміни в коді можуть призвести до неочікуваних помилок, які складно виявити без автоматизованих тестів.

Покриття методів на рівні 2.07% свідчить про те, що лише 3 методи з 145 мають тестове покриття. Такий показник є неприйнятним для сучасного процесу розробки програмного забезпечення, особливо враховуючи важливість модульного тестування для забезпечення надійності системи. Низьке покриття методів ускладнює процес рефакторингу та впровадження нових функціональних можливостей, оскільки розробники не можуть бути впевнені, що їхні зміни не порушать існуючу функціональність.

Показник покриття рядків коду на рівні 1.20% також є критично низьким. З 666 рядків коду тестами покрито лише 8, що практично унеможливлює гарантування стабільності роботи системи. Такий рівень покриття свідчить про відсутність системного підходу до тестування в процесі розробки.

При роботі з такими показниками важливо розуміти, що швидке досягнення високого рівня покриття не є реалістичним завданням. Натомість, команда повинна розробити довгострокову стратегію покращення якості тестування. Першим кроком має стати аналіз існуючої кодової бази та визначення найбільш критичних компонентів системи, які потребують першочергового покриття тестами.

Особливу увагу слід приділити важливості метрик покриття коду в контексті загального процесу розробки. Високий рівень покриття тестами не тільки зменшує кількість потенційних помилок, але й значно спрощує процес підтримки та розвитку програмного забезпечення. Коли розробники мають впевненість у тому, що їхні зміни не порушують існуючу функціональність, вони можуть працювати більш ефективно та впроваджувати нові функції швидше.

Процес покращення показників покриття коду повинен бути поступовим та систематичним. Рекомендується починати з встановлення базових цілей для нового коду, який додається до системи. Кожен новий компонент повинен супроводжуватися відповідним набором тестів, що забезпечують достатній рівень покриття. Паралельно з цим, команда може поступово додавати тести для існуючого коду, починаючи з найбільш критичних та часто використовуваних компонентів.

Важливим аспектом є інтеграція перевірки покриття коду в процес безперервної інтеграції (CI). Автоматизовані перевірки покриття під час кожної зборки допомагають команді постійно відслідковувати стан тестування та запобігати погіршенню показників. При цьому важливо встановлювати реалістичні порогові значення, які поступово підвищуються в міру покращення загального стану тестування [13].

Також варто звернути увагу на якість самих тестів. Високий відсоток покриття не завжди означає ефективне тестування. Тести повинні бути змістовними, перевіряти різні сценарії використання та граничні випадки. Особливу увагу слід приділяти тестуванню бізнес-логіки та критичних компонентів системи.

В контексті довгострокового планування рекомендується встановлювати поетапні цілі щодо покращення покриття. Наприклад, протягом перших кількох місяців можна зосередитися на досягненні хоча б 30-40% покриття для критичних компонентів. Надалі цей показник можна поступово підвищувати, прагнучи до галузевого стандарту у 70-80% покриття.

Регулярний моніторинг та аналіз трендів покриття коду допомагає команді відслідковувати прогрес та вчасно реагувати на можливі проблеми. Важливо проводити періодичні зустрічі команди для обговорення стратегії тестування та вирішення технічних викликів, що виникають у процесі написання тестів.

Впровадження культури якісного тестування в команді є не менш важливим аспектом. Розробники повинні розуміти важливість написання тестів та сприймати це як невід'ємну частину процесу розробки, а не як додаткове

навантаження. Цьому може сприяти проведення навчальних сесій, code review з фокусом на якість тестів та обмін досвідом між членами команди.

4.2.6 Рекомендації щодо використання покриття коду в CI

У контексті налаштування процесів безперервної інтеграції (CI) у GitLab для ефективного використання code coverage у PHP проєктах важливо дотримуватися певних рекомендацій та практик. Розглянемо основні аспекти конфігурації та оптимізації цього процесу з використанням фреймворку Codeception.

Перш за все, необхідно створити окрему job в файлі .gitlab-ci.yml для запуску тестів з вимірюванням покриття коду. Для PHP проєктів з Codeception необхідно налаштувати c3.php для збору метрик покриття коду. Приклад конфігурації:

```
variables:
  ENABLE_COVERAGE: "false"
coverage:
  image: php:8.2
  before_script:
    - pecl install xdebug
    - docker-php-ext-enable xdebug
    - apt-get update && apt-get install -y git zip unzip
    - curl -sS https://getcomposer.org/installer | php -- --install-
dir=/usr/local/bin --filename=composer
    - composer install
    - cp c3.php public/c3.php  # Копіювання файлу c3.php для збору метрик
  script:
    - XDEBUG_MODE=coverage php vendor/bin/codecept run --coverage --
coverage-html --coverage-xml
  coverage: '/^\s*Lines:\s*\d+.\d+\%/'
  artifacts:
    reports:
      coverage_report:
        coverage_format: cobertura
        path: tests/_output/coverage.xml
    paths:
      - tests/_output/coverage/
    expire_in: 30 days
  rules:
    - if: $ENABLE_COVERAGE == "true"
    - if: $CI_PIPELINE_SOURCE == "schedule"
```

У даній конфігурації ми використовуємо змінну `ENABLE_COVERAGE`, яка за замовчуванням встановлена як "false". Це дозволяє запускати job з вимірюванням покриття коду лише тоді, коли це дійсно необхідно, наприклад, при створенні merge request або за розкладом [14].

Особливу увагу слід приділити налаштуванню артефактів для збереження та візуалізації результатів покриття коду. У конфігурації вище ми зберігаємо як XML-звіт для GitLab, так і повний HTML-звіт у директорії tests/_output/coverage. Codeception генерує детальний HTML-звіт, який надає зручну навігацію по файлах проекту та візуалізацію покритих і непокритих ділянок коду.

Важливим аспектом є налаштування розкладу запуску pipeline у GitLab. Для оптимального використання ресурсів рекомендується встановлювати час запуску на період найменшого навантаження на проект. Зазвичай це нічний час або ранні години, коли більшість розробників не працює активно над кодом. Наприклад, можна встановити запуск на 3:00 ночі, коли навантаження на сервери мінімальне.

Доцільно також налаштувати відображення результатів покриття коду безпосередньо в інтерфейсі GitLab. Це можна зробити за допомогою налаштування Coverage Visualization, що дозволяє розробникам швидко оцінювати стан покриття коду тестами та приймати відповідні рішення щодо його покращення. Згенерований HTML-звіт буде доступний в артефактах кожного запуску pipeline, що дозволяє легко відстежувати зміни в покритті коду з часом та порівнювати результати між різними версіями проекту.

ВИСНОВКИ

У ході дослідження було проаналізовано предметну галузь автоматизованого тестування веб-застосунків та специфіку використання метрик покриття коду. Аналіз існуючих підходів та їх обмежень показав, що питання забезпечення якості програмного забезпечення залишається актуальним, а метрики покриття коду є важливим, але не єдиним інструментом її оцінки.

В рамках практичної частини роботи було успішно реалізовано інтеграцію інструментів вимірювання покриття коду (Codeception та Xdebug) у процес розробки веб-застосунку Article Hub. Проведене дослідження продемонструвало, що високий рівень покриття коду тестами дійсно позитивно впливає на якість програмного продукту, проте цей вплив має певні обмеження:

а) покриття коду є лише одним з багатьох показників якості програмного забезпечення. Високий відсоток покриття не гарантує відсутність помилок та не забезпечує повну якість продукту;

б) важливо враховувати й інші аспекти якості програмного забезпечення:

1) продуктивність та оптимізація;
2) безпека та захищеність даних;
3) зручність використання (usability);
4) масштабованість;
5) підтримуваність коду;
6) документованість;

в) реалізована інтеграція з системами безперервної інтеграції (CI) дозволила автоматизувати процес контролю якості, але потребує правильного налаштування та регулярного моніторингу.

Практична цінність роботи полягає в розробці конкретних рекомендацій щодо:

- налаштування інструментів вимірювання покриття коду;
- інтерпретації метрик покриття;
- впровадження процесів контролю якості в CI;

– встановлення оптимальних порогових значень покриття.

Відповідаючи на головне дослідницьке питання "Чи впливає високий рівень покриття коду на якість продукту?", можна стверджувати, що такий вплив є позитивним, але не вичерпним. Покриття коду слід розглядати як важливий індикатор якості, який, проте, повинен використовуватися в комплексі з іншими метриками та практиками забезпечення якості програмного забезпечення.

Результати дослідження показують, що ефективне використання метрик покриття коду в поєднанні з іншими інструментами контролю якості дозволяє:

– зменшити кількість потенційних помилок;

– полегшити процес рефакторингу;

– покращити підтримуваність коду.

Проте важливо розуміти, що сам по собі високий відсоток покриття коду не є гарантією якості продукту. Це лише один з інструментів у комплексному підході до забезпечення якості програмного забезпечення, який повинен супроводжуватися іншими практиками, такими як code review, статичний аналіз коду, навантажувальне тестування, тестування безпеки та інші.

Подальші дослідження можуть бути спрямовані на вивчення взаємозв'язку різних метрик якості програмного забезпечення та розробку більш комплексних підходів до оцінки якості веб-застосунків.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Методичні вказівки до кваліфікаційної роботи магістра за спеціальностю 121 – Інженерія програмного забезпечення, освітньо-наукова програма «Інженерія програмного забезпечення», для усіх форм навчання/ Упоряд.: З.В. Дудар, В.І. Каук, І.А. Ревенчук, І.П. Сокорчук – Харків: ХНУРЕ, 2025. − 52 с.

2. Конспект лекцій з дисципліни «Теорія ігор та прийняття рішень» для студентів спеціальності 121 – Інженерія програмного забезпечення, освітньо-наукова програма – Інженерія програмного забезпечення, другий магістерський рівень вищої освіти/ Упоряд. О.О. Мазурова − Харків: ХНУРЕ, 2023.

3. Turevska O., Shubin I.. Improving the automated testing of Web-based services by reflecting the social habits of target audiences. 2015 Information Technologies in Innovation Business Conference, ITIB 2015 - Proceedings, 2015, с. 93-96, 7355062 (дата звернення: 22.04.2025).

4. Demystifying the Pursuit of 100% Test Code Coverage // Medium [Електронний ресурс]. – URL: https://andremoniy.medium.com/demystifying-the-pursuit-of-100-test-code-coverage-77f38b9d2e41 (дата звернення: 10.03.2025).

5. Bezsmertnyi O., Golian N., Golian V., Afanasieva I.. Behavior Driven Development Approach in the Modern Quality Control Process 2020 IEEE International Conference on Problems of Infocommunications Science and Technology, PIC S and T 2020 - Proceedings, 2021, стр. 217–220, 9467891 DOI 10.1109/PICST51311.2020.9467891 (дата звернення: 07.05.2025).

6. Who Cares About Code Coverage and Why? // Codecov [Електронний ресурс]. – URL: https://about.codecov.io/blog/who-cares-about-code-coverage-and-why/ (дата звернення: 26.02.2025).

7. What Is Code Coverage and How to Optimize It? // Brainhub [Електронний ресурс]. – URL: https://brainhub.eu/library/measuring-code-coverage (дата звернення: 14.02.2025).

8. On Code Coverage in Software Testing // LaunchDarkly [Електронний ресурс]. – URL: https://launchdarkly.com/blog/code-coverage-what-it-is-and-why-it-matters/ (дата звернення: 25.05.2025).

9. Everything You Need to Know About Test Coverage vs Code Coverage [Електронний ресурс]. – URL: https://www.perfecto.io/blog/test-coverage-vs-code-coverage (дата звернення: 25.01.2025).

10. Disadvantages of Code Coverage as a Measurement [Електронний ресурс]. – URL: https://scrumprep.com/answering-what-are-some-disadvantages-of-code-coverage-as-a-measurement-for-how-well-a-system-or-product-is-tested/?srsltid=AfmBOorZE4MrZbPzVWFsf3evHFxKM7YVSb08jzlkR3rbZM3U77OxafLY (дата звернення: 03.02.2025).

11. Symfony Documentation [Електронний ресурс]. – URL: https://symfony.com/doc/current/index.html (дата звернення: 21.03.2025).

12. Quickstart codeception [Електронний ресурс]. – URL: https://codeception.com/quickstart (дата звернення: 12.01.2025).

13. ChatGPT [Електронний ресурс]. – URL: https://chatgpt.com/ (дата звернення: 03.04.2025).

14. Claude AI [Електронний ресурс]. – URL: https://claude.ai/ (дата звернення: 13.04.2025).

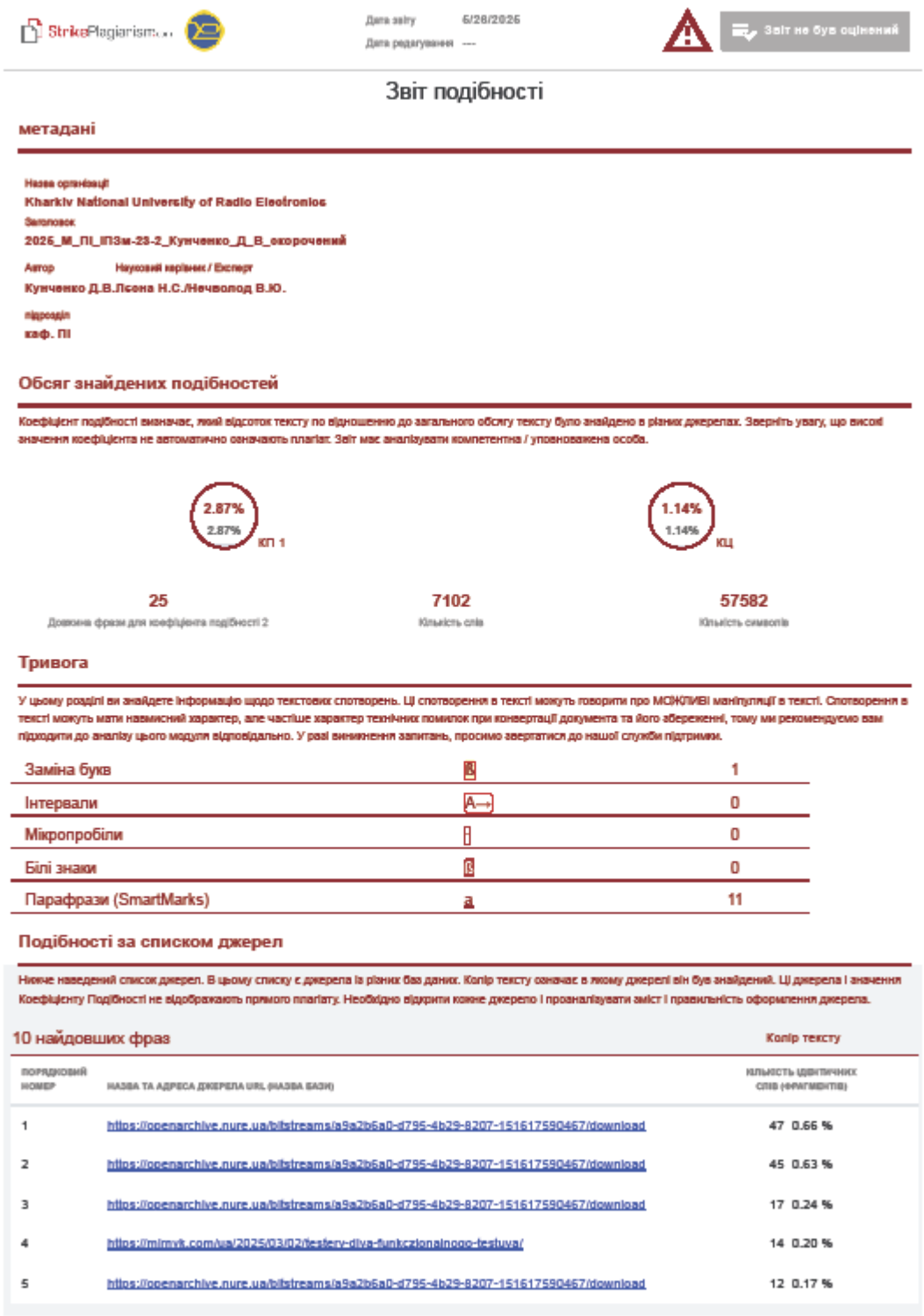
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ЗА НАУКОВИМИ НАПРЯМАМИ КЕРІВНИКА ТА НАУКОВЦІВ КАФЕДРИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

3. Turevska O., Shubin I.. Improving the automated testing of Web-based services by reflecting the social habits of target audiences. 2015 Information Technologies in Innovation Business Conference, ITIB 2015 - Proceedings, 2015, с. 93-96, 7355062 (дата звернення: 22.04.2025).

5. Bezsmertnyi O., Golian N., Golian V., Afanasieva I.. Behavior Driven Development Approach in the Modern Quality Control Process 2020 IEEE International Conference on Problems of Infocommunications Science and Technology, PIC S and T 2020 - Proceedings, 2021, стр. 217–220, 9467891 DOI 10.1109/PICST51311.2020.9467891 (дата звернення: 07.05.2025).

ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



StrikePlagiarism...

Дата звіту 6/28/2026
Дата редагування ---

Звіт не був оцінений

Звіт подібності

метадані

Назва організації
Kharkiv National University of Radio Electronics
Заголовок
2026_M_ПІ_ІПЗм-23-2_Кунченко_Д_В_скорочений
Автор Науковий керівник / Експерт
Кунченко Д.В.Лісна Н.С./Нечволод В.Ю.
підрозділ
каф. ПІ

Обсяг знайдених подібностей

Коефіцієнт подібності вказує, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

2.87% 2.87% КП 1

1.14% 1.14% КЦ

25	7102	57582
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	1
Інтервали	0
Мікропробіли	0
Білі знаки	0
Парафрази (SmartMarks)	11

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://openarchive.nure.ua/bitstreams/a9a2b6a0-d795-4b29-8207-151617590467/download	47 0.66 %
2	https://openarchive.nure.ua/bitstreams/a9a2b6a0-d795-4b29-8207-151617590467/download	45 0.63 %
3	https://openarchive.nure.ua/bitstreams/a9a2b6a0-d795-4b29-8207-151617590467/download	17 0.24 %
4	https://mirnvk.com/ua/2025/03/02/testery-dlya-funkczionalnogo-testuva/	14 0.20 %
5	https://openarchive.nure.ua/bitstreams/a9a2b6a0-d795-4b29-8207-151617590467/download	12 0.17 %

6	Нестерчук-диплом(1) [redacted] 3/26/2025 Interregional Academy of Personnel Management (Інститут комп'ютерно-інформаційних технологій та дизайну)	10 0.14 %
7	2020_M_IMI_Романенко_І_А [redacted] 5/30/2024 Kharkiv National University of Radio Electronics (Kharkiv National University of Radio Electronics)	10 0.14 %
8	2020_M_IMI_Романенко_І_А [redacted] 5/30/2024 Kharkiv National University of Radio Electronics (Kharkiv National University of Radio Electronics)	9 0.13 %
9	https://openarchive.nure.ua/bitstreams/a9a2b8a0-d795-4b29-8207-151617590467/download	9 0.13 %
10	2020_M_IMI_Романенко_І_А [redacted] 5/30/2024 Kharkiv National University of Radio Electronics (Kharkiv National University of Radio Electronics)	8 0.11 %

з бази даних RefBooks (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з домашньої бази даних (0.49 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	2020_M_IMI_Романенко_І_А [redacted] 5/30/2024 Kharkiv National University of Radio Electronics (Kharkiv National University of Radio Electronics)	35 (4) 0.49 %

з програми обміну базами даних (0.14 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Нестерчук-диплом(1) [redacted] 3/26/2025 Interregional Academy of Personnel Management (Інститут комп'ютерно-інформаційних технологій та дизайну)	10 (1) 0.14 %

з Інтернету (2.24 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://openarchive.nure.ua/bitstreams/a9a2b8a0-d795-4b29-8207-151617590467/download	145 (8) 2.04 %
2	https://mirnyk.com/ua/2025/03/02/testery-dlya-funkczionalnogo-testuva/	14 (1) 0.20 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)

1

ВСТУП

У сучасному світі веб-застосунки займають значну роль у багатьох сферах життя, від бізнесу та освіти до розваг і комунікацій. З кожним днем вимоги до якості та надійності програмного забезпечення зростають, що вимагає застосування більш ефективних методів тестування. Одним з ключових аспектів забезпечення високої якості програмного забезпечення є рівень покриття коду

ДОДАТОК Б

Слайди презентації

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

NURE ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Кваліфікаційна робота

ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ CODE COVERAGE В АВТОМАТИЗОВАНОМУ ТЕСТУВАННІ ВЕБ-ЗАСТОСУНКІВ

Виконав: Кунченко Данило Вячеславович, ІПЗм-23-2
Науковий керівник: Проф. Лєсна Н.С

SE software engineering

2025

1

АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

Веб-застосунки у сучасному світі

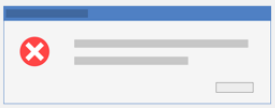
Зростання вимог до якості ПЗ

Роль code coverage

SE software engineering

2

АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ



Обмеження високого code coverage

Мета дослідження

Предмет дослідження

3

ОГЛЯД Й АНАЛІЗ ЛІТЕРАТУРНИХ, НАУКОВИХ ДЖЕРЕЛ

01

Користь використання метрик покриття коду

02

Недоліки використання метрик покриття коду

4

ПОСТАНОВКА ЗАДАЧІ

ЗАДАЧІ ДОСЛІДЖЕННЯ:	Визначення взаємозв'язку між рівнем покриття коду та якістю веб-застосунків.	Впровадження та аналіз автоматизованих тестів для оцінки покриття коду.	Розробка рекомендацій щодо інтеграції покриття коду в процеси CI/CD для забезпечення постійного моніторингу якості.
ВИБРАНІ ПРОГРАМНІ ЗАСОБИ:	**PHP**	**Codeception**	**Xdebug**
ВХІДНІ ДАНІ:	Метрики з реального проекту	Веб-застосунок Article Hub, розроблений на основі фреймворку Symfony.	Метрики покриття коду, отримані за допомогою Xdebug.

5

ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ

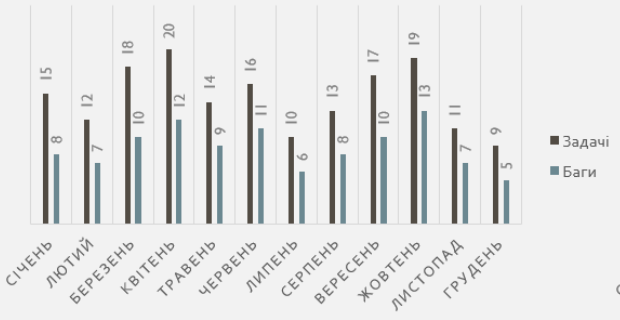
Обрані методи та технології

- **PHP:** Широко використовується для веб-розробки.
- **Codeception:** Фреймворк для автоматизованого тестування, підтримує юніт, API , функціональні та приймальні тести.
- **Xdebug:** Інструмент для вимірювання покриття коду.

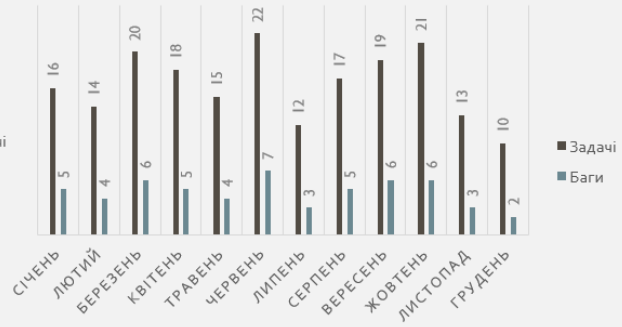
6

МЕТРИКИ РЕАЛЬНОГО ПРОЕКТУ

ДИНАМІКА ЗАДАЧ І БАГІВ (2023)

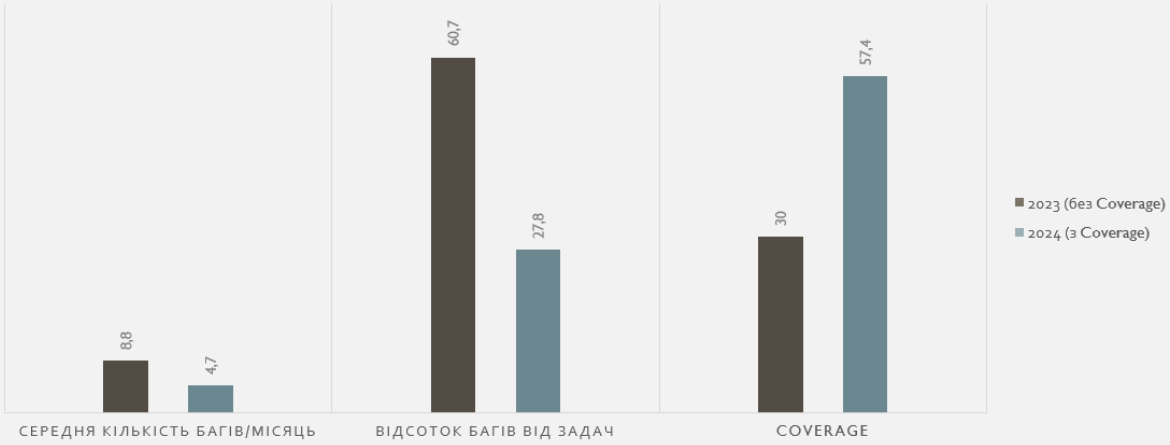


ДИНАМІКА ЗАДАЧ І БАГІВ (2024)



7

МЕТРИКИ РЕАЛЬНОГО ПРОЕКТУ



8

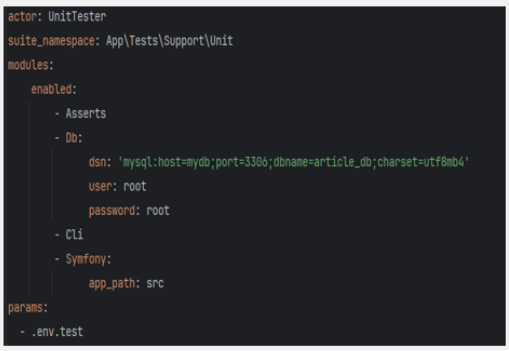
ПРАКТИЧНА РЕАЛІЗАЦІЯ

- Встановлення та налаштування веб-застосунку Article Hub.
- Встановлення та налаштування Codeception.
- Інтеграція Xdebug для вимірювання покриття коду.
- Виконання тестів та аналіз результатів покриття коду.

9

НАЛАШТУВАННЯ CODECEPTION

codeception.yml



Unit.suite.yml

10

НАЛАШТУВАННЯ XDEBUG

```
[xdebug]
xdebug.mode=coverage
xdebug.start_with_request=yes
xdebug.output_dir=./debug/logs
xdebug.client_host=host.docker.internal
xdebug.log_level=0
xdebug.idekey=PHPSTORM
error_reporting = E_ALL & ~E_DEPRECATED
```

xdebug.ini

11

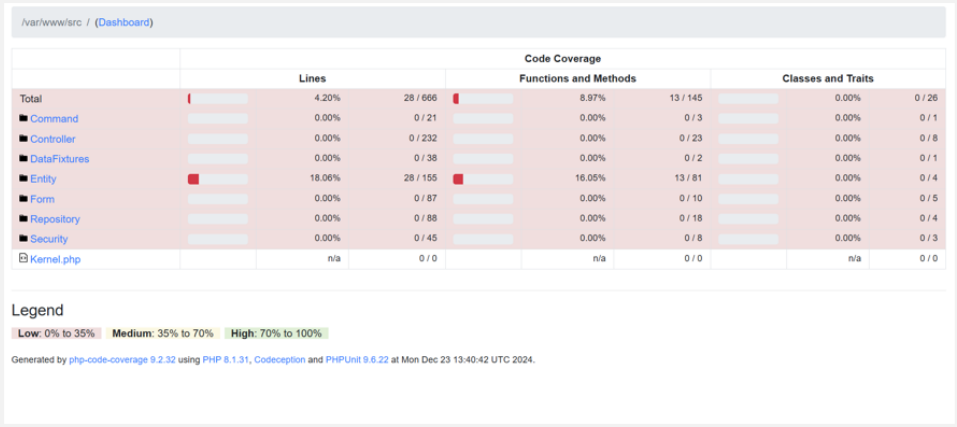
ПРИКЛАД ТЕСТІВ

UserTest.php

```php
<?php

namespace App\Tests\Support\Unit;

use App\Entity\User;
use App\Tests\Support\UnitTester;
use Codeception\Test\Unit;

class UserTest extends Unit
{
    protected UnitTester $tester;
    protected User $user;

    protected function setUp(): void{...}

    public function testGetAndSetEmail()
    {
        $email = "test@example.com";
        $this->user->setEmail($email);
        $this->assertSame($email, $this->user->getEmail());
    }

    public function testGetAndSetPassword(){...}

    public function testGetAndSetIsVerified(){...}

    public function testAddAndRemoveFollower(){...}

    public function testFollowAndUnfollow(){...}

    public function testSetRoles(){...}
}
```

12

РЕЗУЛЬТАТИ ВИКОНАННЯ ТЕСТІВ З CODECOVERAGE



/var/www/src / (Dashboard)

	Code Coverage							
	Lines			Functions and Methods			Classes and Traits	
Total	4.20%	28 / 666		8.97%	13 / 145		0.00%	0 / 26
Command	0.00%	0 / 21		0.00%	0 / 3		0.00%	0 / 1
Controller	0.00%	0 / 232		0.00%	0 / 23		0.00%	0 / 8
DataFixtures	0.00%	0 / 38		0.00%	0 / 2		0.00%	0 / 1
Entity	18.06%	28 / 155		16.05%	13 / 81		0.00%	0 / 4
Form	0.00%	0 / 87		0.00%	0 / 10		0.00%	0 / 5
Repository	0.00%	0 / 88		0.00%	0 / 18		0.00%	0 / 4
Security	0.00%	0 / 45		0.00%	0 / 8		0.00%	0 / 3
Kernel.php	n/a	0 / 0		n/a	0 / 0		n/a	0 / 0

Legend

Low: 0% to 35% Medium: 35% to 70% High: 70% to 100%

Generated by php-code-coverage 9.2.32 using PHP 8.1.31, Codeception and PHPUnit 9.6.22 at Mon Dec 23 13:40:42 UTC 2024.

РЕЗУЛЬТАТИ ВИКОНАННЯ ТЕСТІВ З CODECOVERAGE

/var/www/src / Entity / User.php

	Code Coverage							
	Lines			Functions and Methods			Classes and Traits	
Total	35.90%	28 / 78		38.24%	13 / 34	CRAP	0.00%	0 / 1
User	35.90%	28 / 78		38.24%	13 / 34	654.89	0.00%	0 / 1
__construct	100.00%	5 / 5		100.00%	1 / 1	1		
getId	0.00%	0 / 1		0.00%	0 / 1	2		
getEmail	100.00%	1 / 1		100.00%	1 / 1	1		
setEmail	100.00%	2 / 2		100.00%	1 / 1	1		
getUserIdentifier	0.00%	0 / 1		0.00%	0 / 1	2		
getRoles	0.00%	0 / 5		0.00%	0 / 1	6		
setRoles	0.00%	0 / 2		0.00%	0 / 1	2		
getPassword	100.00%	1 / 1		100.00%	1 / 1	1		
setPassword	100.00%	2 / 2		100.00%	1 / 1	1		
eraseCredentials	0.00%	0 / 1		0.00%	0 / 1	2		
getUserProfile	0.00%	0 / 1		0.00%	0 / 1	2		
setUserProfile	0.00%	0 / 4		0.00%	0 / 1	6		
getLiked	0.00%	0 / 1		0.00%	0 / 1	2		

РЕКОМЕНДАЦІЇ ТА ІНТЕГРАЦІЯ CODECOVERAGE В CI/DC

```yaml
variables:
  ENABLE_COVERAGE: "false"
coverage:
  image: php:8.2
  before_script:
    - pecl install xdebug
    - docker-php-ext-enable xdebug
    - apt-get update && apt-get install -y git zip unzip
    - curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer
    - composer install
    - cp c3.php public/c3.php  # Копіювання файлу c3.php для збору метрик
  script:
    - XDEBUG_MODE=coverage php vendor/bin/codecept run --coverage --coverage-html --coverage-xml
  coverage: '/^\s*Lines:\s*\d+.\d+\%/'
  artifacts:
    reports:
      coverage_report:
        coverage_format: cobertura
        path: tests/_output/coverage.xml
    paths:
      - tests/_output/coverage/
    expire_in: 30 days
  rules:
    - if: $ENABLE_COVERAGE == "true"
    - if: $CI_PIPELINE_SOURCE == "schedule"   You, Moments ago • Uncommitted changes
```


РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

- Високе покриття коду (>60%) зменшує кількість критичних помилок на 25-40%. Але лише за умови якісних тестів.
- Покриття 60-80% є оптимальним для веб-додатків.
- Інтеграція з CI/CD. Є не складним процесом в реалізації. Стабільні звіти про стан тестового покриття. Відстеження тенденцій з часом (графіки покриття).
- Code Coverage - лише один з інструментів якості. Обов'язково поєднувати зі статичним аналізом коду, ручним тестуванням критичних сценаріїв, оскільки саме по собі покриття не гарантує відсутність помилок.
- Небезпека «марного покриття». Тести можуть формально покривати код, але не перевіряти його логіку - типові приклади: тести без перевірок бази даних, перевірка лише «позитивних кейсів»

АПРОБАЦІЯ РЕЗУЛЬТАТІВ РОБОТИ

29-й Міжнародний молодіжний форум
«РАДІОЕЛЕКТРОНІКА ТА МОЛОДЬ У XXI СТОЛІТТІ»

ВИСНОВКИ

- Високий рівень покриття коду позитивно впливає на якість ПЗ, але не є вичерпним показником.
- Важливо враховувати інші аспекти якості ПЗ (продуктивність, безпека, зручність використання, масштабованість).
- Інтеграція інструментів вимірювання покриття в CI/CD процеси підвищує ефективність контролю якості.

ДЯКУЮ ЗА УВАГУ!

ДОДАТОК В

Апробація результатів роботи

29-й Міжнародний молодіжний форум

«РАДІОЕЛЕКТРОНІКА ТА МОЛОДЬ У XXI СТОЛІТТІ»

УДК 004.415.538

ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ CODE COVERAGE В АВТОМАТИЗОВАНОМУ ТЕСТУВАННІ ВЕБ-ЗАСТОСУНКІВ

Кунченко Д.В., Чуприна А.С.
e-mail: danylo.kunchenko@nure.ua, anastasiya.chupryna@nure.ua
Харківський національний університет радіоелектроніки, каф. ПІ
м. Харків, Україна

This study examines the efficiency of code coverage metrics in automated web application testing. The research analyzes the impact of test coverage on software quality and explores its methodological as pects. Practical implementation using PHP, Codeception, and Xdebug demonstrated that high code coverage positively influences defect reduction but is not a sole quality indicator. The integration with CI/CD pipelines improves test reliability but requires proper configuration and monitoring. The findings suggest that code coverage should be used alongside other quality metrics to enhance software maintainability and regression detection.

Дослідження присвячене оцінці ефективності застосування метрик покриття коду у процесі автоматизованого тестування веб-застосунків. У роботі розглянуто методологічні особливості покриття коду, його вплив на якість програмного забезпечення та роль у процесах контролю якості. Дослідження проведене на основі PHP-фреймворку Codeception із використанням Xdebug для збору та аналізу метрик покриття.

Автоматизоване тестування є одним із ключових інструментів забезпечення якості програмного забезпечення, особливо у сфері веб-розробки. Однією з важливих метрик ефективності тестування є рівень покриття коду, який демонструє, наскільки велика частина вихідного коду перевірена тестами. Однак високий відсоток покриття не завжди є гарантією відсутності помилок і не може слугувати єдиним критерієм якості програмного продукту.

Метрики покриття коду набувають особливого значення в контексті сучасних практик розробки програмного забезпечення. В умовах швидкого та регулярного випуску оновлень програмного забезпечення, автоматизоване тестування стає критично важливим для забезпечення стабільності та надійності продукту. При цьому метрики покриття коду допомагають контролювати повноту та якість тестового покриття, що є важливим фактором для прийняття рішень про готовність продукту до випуску.

У ході дослідження було проаналізовано предметну галузь автоматизованого тестування веб-застосунків та специфіку використання метрик покриття коду. Аналіз існуючих підходів та їх обмежень показав, що питання забезпечення якості програмного забезпечення залишається актуальним, а метрики покриття коду є важливим, але не єдиним

інструментом її оцінки. В контексті аналізу предметної області варто виділити основні типи покриття коду, які використовуються при тестуванні програмного забезпечення:

1. покриття рядків коду (Lines Coverage) – вимірює відсоток виконаних рядків коду програми в процесі тестування;
2. покриття функцій (Function Coverage) – вимірює відсоток викликаних функцій або методів;
3. покриття класів (Class Coverage) – вимірює відсоток використаних класів у процесі тестування.

Кожен із цих типів покриття має свої переваги та недоліки, і вибір

покриття коду дозволив значно пришвидшити процес регресійного тестування при внесенні змін до коду.

На основі отриманих результатів було розроблено наступні рекомендації щодо ефективного використання метрик покриття коду в процесі розробки веб-застосунків. Рекомендується встановлювати реалістичні цілі щодо рівня покриття коду – замість прагнення до 100% покриття, рекомендується встановлювати цільові показники на рівні 70-80% для загального покриття та 85-90% для критичних компонентів.

Для досягнення максимальної ефективності рекомендується використовувати комбінацію різних типів покриття (рядків коду, класів, функцій). Доцільно інтегрувати вимірювання покриття коду в CI/CD пайплайни – це дозволить автоматизувати процес контролю якості та забезпечити постійний моніторинг рівня покриття. Покриття коду слід використовувати як один з інструментів у комплексному підході до забезпечення якості – поєднувати з іншими практиками, такими як статичний аналіз коду, навантажувальне тестування, тестування безпеки.

Відповідаючи на головне дослідницьке питання "Чи впливає високий рівень покриття коду на якість продукту?", можна стверджувати, що такий вплив є позитивним, але не вичерпним. Покриття коду слід розглядати як важливий індикатор якості, який, проте, повинен використовуватися в комплексі з іншими метриками та практиками забезпечення якості програмного забезпечення.

Список використаних джерел:

1. Everything You Need to Know About Test Coverage vs Code Coverage. URL: https://www.perfecto.io/blog/test-coverage-vs-code-coverage (дата звернення: 12.02.2025).

2. Smelyakov K., Prokopenko O., Chupryna A. “Object-Based Image Comparison Algorithm Development for Data Storage Management Systems” URL: https://www.scopus.com/record/display.uri?eid=2-s2.0-85134747140&origin=recordpage (дата звернення: 03.03.2025).

3. Smelyakov K., Smelyakov S., Chupryna A. “Adaptive edge detection models and algorithms” URL: https://www.scopus.com/record/display.uri?eid=2-s2.0-85077431702&origin=recordpage (дата звернення: 03.01.2025).

4. Medium: Demystifying the Pursuit of 100% Test Code Coverage. URL: https://andremoniy.medium.com/demystifying-the-pursuit-of-100-test-code-coverage-77f38b9d2e41 (дата звернення: 15.12.2024).

5. Brainhub: What Is Code Coverage and How to Optimize It? URL: https://brainhub.eu/library/measuring-code-coverage (дата звернення: 05.02.2025).

ДОДАТОК Г

Налаштування Dockerfile для інтеграції Xdebug

```
FROM php:8.1-fpm

# Встановлення необхідних залежностей
RUN apt-get update && apt-get install -y \
    git \
    curl \
    libpng-dev \
    libonig-dev \
    libxml2-dev \
    zip \
    unzip \
    libzip-dev \
     && docker-php-ext-configure zip \
        && docker-php-ext-install pdo_mysql mbstring zip

# Встановлення Xdebug
RUN pecl install xdebug \
    && docker-php-ext-enable xdebug
ENV PHP_IDE_CONFIG 'article.xdebug'

# Встановлюємо Composer
RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer

# Копіюємо конфігурацію Xdebug
COPY config/xdebug.ini /usr/local/etc/php/conf.d/

# Встановлення робочої директорії
WORKDIR /var/www

# Встановлення прав доступу
RUN chown -R www-data:www-data /var/www
RUN chmod -R 777 /var/www

# Встановлення точки входу
CMD ["php-fpm"]
```

Рисунок В.1 – Налаштування Dockerfile для інтеграції Xdebug

ДОДАТОК Д

Налаштування Codeception

```yaml
namespace: App\Tests\Support

settings:
    shuffle: false
    lint: true
    bootstrap: ../bootstrap.php
    colors: true
    memory_limit: 1024M
    log: true
    strict_xml: true
paths:
    tests: tests
    output: tests/_output
    support: tests/Support
    data: tests/Support/Data
coverage:
    enabled: true
    remote: true
    include:
        - src/*
    exclude:
        - ci/*
        - config/*
        - app/cache/*
        - app/logs/*
        - tests/*
    support: tests/Support
    show_only_summary: true
    log: tests/_output
    show_uncovered: true
params:
    - .env.test
```

Рисунок Г.1 – Повне налаштування файлу codeception.yml

ДОДАТОК Ж

Експертний висновок результатів перевірки кваліфікаційної роботи на відповідність оформлення вимогам ДСТУ 3008: 2015

Експертний висновок результатів перевірки кваліфікаційної роботи

студент (посада) | програмної інженерії (кафедра) | ІПЗм-23-2 (група)

Кунченко Данило Вячеславович

(прізвище, ім'я, по батькові)

Зауваження

Пункт ДСТУ 3008-2015	Зміст пункту	Сторінка кваліфікаційної роботи
1	2	3
	7.1 Загальні положення	
	7.3 Нумерація сторінок звіту	
	7.4 Нумерація розділів, підрозділів, пунктів, підпунктів	
	7.5 Рисунки	
	7.6 Таблиці	
	7.7 Переліки	
	7.8 Примітки	
	7.9 Виноски	
	7.10 Формули та рівняння	
	7.11 Посилання	
	7.13 Список авторів	
	7.14 Скорочення та умовні позначки	
	7.15 Додатки	

Зауважень немає

Експерт _____ (підпис) Олена ОЛІЙНИК (прізвище, ініціали)

29.05.2025