НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет/(ННІ) **Інформаційних технологій**

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету (Директор ННІ)	**Завідувач кафедри**
(назва факультету (ННІ))	(назва кафедри)
(підпис) Ігор Болбот (ім'я ПРІЗВИЩЕ)	(підпис) Белла Голуб (ім'я ПРІЗВИЩЕ)
"___" _____ 2025р.	"___" _____ 2025р.

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему Система аналізу результатів інформаційної підтримки тестування та сертифікації

Спеціальність 121 «Інженерія програмного забезпечення»
(код і найменування)

Освітня програма Програмне забезпечення інформаційних систем
(назва)

Орієнтація освітньої програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Гарант освітньої програми

к.ф-м.н. доцент _____ Віктор КИРИЧЕНКО
(науковий ступінь та вчене звання) (підпис) (ім'я ПРІЗВИЩЕ)

Керівник магістерської кваліфікаційної роботи

к.т.н. доцент _____ Роман ПОНОМАРЕНКО
(науковий ступінь та вчене звання) (підпис) (ім'я ПРІЗВИЩЕ)

Виконав _____ Роман ХОМЕНКО
(підпис) (ім'я ПРІЗВИЩЕ здобувача)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ

І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет (ННІ) _____**Інформаційних технологій**_____

ЗАТВЕРДЖУЮ
Завідувач кафедри
____комп'ютерних наук _____
__.т.н., доцент__ _____ _Белла ГОЛУБ_
(науковий ступінь, вчене звання) (підпис) ((ім'я ПРІЗВИЩЕ))
"____" _____20_____року

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

_____Хоменко Роман Миколайович_____
(прізвище, ім'я, по батькові)
Спеціальність _____121 «Інженерія програмного забезпечення»_____
(код і найменування)
Освітня програма _____Програмне забезпечення інформаційних систем_____
(назва)
Орієнтація освітньої програми_____освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)
Тема магістерської кваліфікаційної роботи_____
__Система аналізу результатів інформаційної підтримки тестування та сертифікації___
затверджена наказом від "_10_" __жовтня_2025 р. №__2291 "С"__

Термін подання завершеної роботи на кафедру _____
(рік, місяць, число)
Вихідні дані до магістерської кваліфікаційної роботи _____
__ Результати автоматизованих тестів_____
Перелік питань, що підлягають дослідженню:

1. Як виявляти та маркувати flaky-тести, і як це впливає на метрики та підсумкові вердикти?
2. Наскільки запропонований конвеєр скорочує час підготовки звіту та підвищує точність/повноту виявлення невідповідностей порівняно з baseline?
3. Які ризики та загрози валідності (неповні дані, зсуви конфігурацій, людський фактор) і як їх пом'якшити?
4. Які вимоги до продуктивності й масштабованості потрібні для щоденних/релізних оцінок?
5. Які механізми аудиту та версіонування потрібні для доказовості сертифікації?

Перелік графічного матеріалу (за потреби) _____
_____

Дата видачі завдання "__01__"___листопада____2025 р.

Керівник магістерської кваліфікаційної роботи _____ _Р.М. ПОНОМАРЕНКО__
(підпис) (ім'я ПРІЗВИЩЕ)

Завдання прийняв до виконання _____ __Роман ХОМЕНКО___
(підпис) (ім'я ПРІЗВИЩЕ)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

- → — відображення/посилання (напр., «тест → результат»)
- ↔ — взаємно-однозначний зв’язок / трасованість (напр., «вимога ↔ тест»)
- **SDLC** — Software Development Life Cycle (життєвий цикл ПЗ)
- **V&V** — Verification & Validation (верифікація і валідація)
- **QA/QE** — Quality Assurance / Quality Engineering
- **CI/CD** — Continuous Integration / Continuous Delivery
- **DWH** — Data Warehouse (сховище даних)
- **OLTP/OLAP** — Online Transaction/Analytical Processing
- **API** — Application Programming Interface
- **UUID/ULID** — універсальні унікальні ідентифікатори
- **SHA** — Secure Hash Algorithm (хеш коміту/артефакту)
- **RBAC** — Role-Based Access Control (рольовий доступ)

ВСТУП

Сучасні інформаційні системи охоплюють практично всі сфери людської діяльності — від фінансів і охорони здоров’я до освіти та державного управління. Зростання складності програмного забезпечення призводить до підвищення вимог до його якості, безпеки, надійності та відповідності міжнародним стандартам. У зв’язку з цим тестування та сертифікація програмних продуктів стають невід’ємними етапами життєвого циклу програмного забезпечення.

Процес сертифікації ПЗ передбачає підтвердження того, що продукт відповідає встановленим критеріям якості та безпеки. Для цього проводиться велика кількість тестів, результати яких необхідно правильно обробити, проаналізувати й інтерпретувати. Проте у сучасних умовах зростає обсяг даних, отриманих під час тестування, що робить ручний аналіз результатів малоефективним і схильним до помилок.

Актуальність дослідження полягає у зростаючій потребі в автоматизації процесів тестування та сертифікації програмного забезпечення. Зі збільшенням обсягів і складності ПЗ традиційні методи перевірки якості стають малоефективними. Створення системи, яка дозволяє аналізувати результати тестування, формувати аналітичні звіти та підтримувати прийняття рішень щодо сертифікації, сприятиме підвищенню об’єктивності оцінювання, скороченню часу перевірки та покращенню якості програмних продуктів.

Мета дослідження – проаналізувати та обґрунтувати підходи до побудови системи аналізу результатів інформаційної підтримки тестування та сертифікації програмного забезпечення, дослідити методи оцінювання якості та ефективності тестування, а також визначити шляхи підвищення об’єктивності та автоматизації процесу сертифікації.

Об'єкт дослідження – процес тестування та сертифікації програмного забезпечення, який включає перевірку його функціональності, якості, надійності та відповідності встановленим стандартам.

Предмет дослідження – методи, моделі та засоби аналізу результатів тестування програмного забезпечення для забезпечення інформаційної підтримки процесу сертифікації.

Завдання дослідження:

- Проаналізувати сучасні підходи та стандарти, що регламентують процеси тестування і сертифікації програмного забезпечення.
- Дослідити існуючі методи обробки та аналізу результатів тестування.
- Визначити вимоги до системи аналізу результатів інформаційної підтримки сертифікації.
- Розробити концептуальну модель або алгоритм аналізу результатів тестування для підтримки сертифікаційного процесу.
- Провести оцінку ефективності запропонованого підходу та сформулювати практичні рекомендації щодо його застосування.

Для досягнення мети використані такі *методи* дослідження:

- *Аналіз науково-технічної літератури та нормативних документів* – для вивчення стандартів тестування та сертифікації ПЗ.
- *Системний аналіз* – для вивчення процесів тестування, структури та взаємозв'язків даних.
- *Моделювання та алгоритмізація* – для побудови концептуальної моделі обробки результатів тестування.

- *Статистичний аналіз* – для обробки тестових даних та оцінки показників якості ПЗ.
- *Метод експертних оцінок* – для визначення критеріїв відповідності стандартам та оцінки ефективності підходів.

Дослідження процесів аналізу результатів інформаційної підтримки тестування та сертифікації програмного забезпечення може принести значну користь як розробникам ПЗ, так і організаціям, що здійснюють сертифікацію. Результати дослідження можуть бути використані для підвищення ефективності оцінки якості ПЗ, автоматизації обробки результатів тестування та забезпечення об’єктивності прийняття рішень щодо сертифікації. Система аналізу результатів тестування дозволяє агрегувати дані з різних джерел, оцінювати показники якості та надійності продукту, а також формувати аналітичні звіти для підтримки експертного рішення. Основними перевагами такого підходу є зменшення часу на обробку великих обсягів тестових даних, підвищення точності оцінки відповідності стандартам, виявлення потенційних ризиків і забезпечення прозорості процесу сертифікації.

Дослідження представлено в трьох розділах магістерської роботи. У першому розділі проведено аналіз сучасних підходів до тестування та сертифікації ПЗ, нормативних документів і стандартів якості. У другому розділі змодельовано об’єкт та предмет дослідження та обґрунтовано методи аналізу результатів тестування. У третьому розділі виконано дослідження ефективності запропонованих підходів та розглянуто приклади застосування системи на конкретному програмному продукті.

За результатами роботи отримано висновки щодо ефективності методів аналізу результатів тестування та рекомендації щодо їх застосування для підвищення об’єктивності та автоматизації процесів сертифікації програмного забезпечення.

Кваліфікаційна робота викладена на 64 сторінці, вона містить 3 розділи, 8 ілюстрацій, 4 таблиці, 10 джерел в списку використаних джерел.

РОЗДІЛ 1

ПРЕДМЕТНА ОБЛАСТЬ ТЕСТУВАННЯ ТА СЕРТИФІКАЦІЇ ПЗ

1.1 Загальна характеристика тестування та сертифікації ПЗ

Тестування програмного забезпечення є невід'ємною складовою життєвого циклу будь-якого ПЗ та має на меті перевірку його відповідності функціональним і нефункціональним вимогам. Воно забезпечує виявлення дефектів на різних етапах розробки, підвищує надійність продукту та зменшує ризики виникнення помилок під час експлуатації. Процес тестування включає не лише перевірку правильності виконання функцій, а й оцінку продуктивності, стабільності, безпеки, сумісності з іншими системами та зручності для користувачів (рис. 1.1).

Тестування проводиться на кількох рівнях. Модульне тестування перевіряє окремі компоненти системи на коректність виконання функцій. Інтеграційне тестування спрямоване на оцінку взаємодії модулів та їхньої здатності коректно обмінюватися даними. Системне тестування охоплює перевірку всього ПЗ на відповідність вимогам і стандартам. Прийомне тестування дозволяє оцінити, чи задовольняє продукт потреби користувачів і бізнес-цілі. Додатково проводиться навантажувальне, стрес- та безпекове тестування, що дозволяє визначити поведінку системи в умовах високого навантаження та перевірити її стійкість до потенційних загроз.

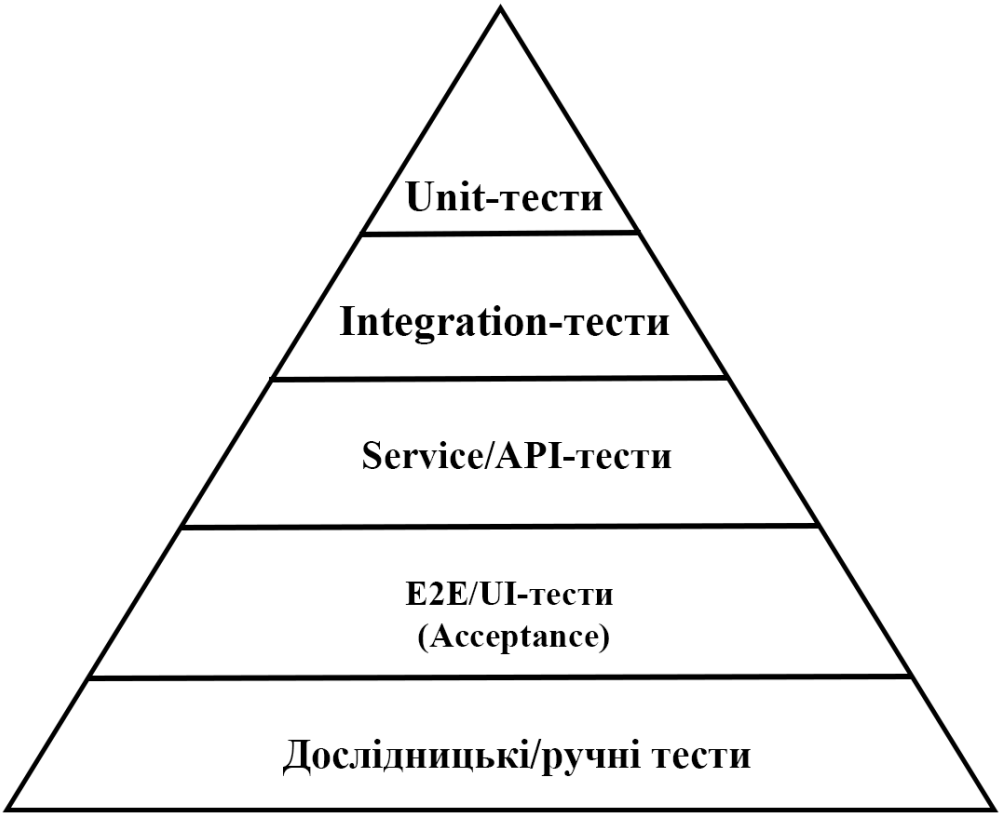

Рисунок 1.1 - Піраміда тестування

Сертифікація програмного забезпечення є процесом формальної оцінки ПЗ компетентними органами або експертами. Вона підтверджує, що продукт відповідає вимогам якості, надійності та безпеки, а також встановленим стандартам і нормативам. Основні рівні сертифікації включають перевірку функціональної відповідності, оцінку нефункціональних характеристик та відповідність міжнародним або галузевим стандартам, таким як ISO/IEC 25010, ISO/IEC 27001, IEC 62304 або Common Criteria (ISO/IEC 15408).

Процес сертифікації передбачає кілька етапів. На першому етапі здійснюється підготовка документації та визначення критеріїв відповідності. Далі проводиться тестування з об’єктивним збором результатів, їхня обробка та аналіз. Завершальний етап включає формування сертифікаційного

висновку, який офіційно підтверджує відповідність продукту вимогам стандартів. Сертифікація не лише підвищує рівень довіри до ПЗ, але й забезпечує об’єктивність оцінки якості, зменшує ризики експлуатаційних проблем та сприяє підвищенню конкурентоспроможності продукту на ринку.

Сучасні умови розробки ПЗ, що характеризуються швидким ростом складності систем, великою кількістю функцій та інтеграцій, підвищують значущість автоматизованих систем аналізу результатів тестування. Використання таких систем дозволяє не тільки збирати та обробляти великі обсяги тестових даних, а й оцінювати відповідність стандартам, прогнозувати потенційні ризики та формувати рекомендації для прийняття рішень щодо сертифікації. Це підвищує ефективність процесу тестування та забезпечує більш точне і своєчасне підтвердження якості та безпеки програмного забезпечення.

Основною метою тестування програмного забезпечення є забезпечення відповідності продукту встановленим вимогам та стандартам якості, а також виявлення і усунення дефектів на ранніх етапах розробки. Тестування дозволяє гарантувати стабільність, надійність та безпеку ПЗ, підвищує ефективність його експлуатації та знижує ризики виникнення критичних помилок під час використання кінцевими користувачами.

До основних цілей тестування належать:

- Перевірка функціональності – оцінка того, чи виконує програмний продукт усі заявлені функції та відповідає технічним і бізнес-вимогам.
- Оцінка надійності та стабільності – визначення здатності ПЗ працювати коректно протягом заданого часу та в різних умовах експлуатації.
- Забезпечення безпеки – виявлення вразливостей, запобігання несанкціонованому доступу та захист даних користувачів.

- Контроль продуктивності – перевірка швидкодії, часу відгуку, здатності витримувати навантаження та ефективності використання ресурсів.
- Перевірка сумісності – оцінка роботи ПЗ у взаємодії з іншими системами, платформами та середовищами виконання.
- Підвищення якості користувацького досвіду – перевірка зручності використання, логіки інтерфейсу та відповідності очікуванням користувачів.

Досягнення цих цілей дозволяє не лише підвищити якість програмного продукту, але й створює основу для об’єктивної сертифікації, оскільки результати тестування формують підґрунтя для оцінки відповідності ПЗ міжнародним та галузевим стандартам.

1.2 Сучасні підходи до аналізу результатів тестування

У сучасних умовах розвитку інформаційних технологій аналіз результатів тестування набуває все більшого значення, оскільки саме на основі тестових даних приймаються рішення про якість, надійність і готовність програмного продукту до експлуатації або сертифікації. Обсяги даних, що генеруються під час автоматизованого тестування, постійно зростають, тому ручна обробка інформації стає неефективною. Це зумовлює необхідність застосування автоматизованих і аналітичних підходів до збору, оцінки та візуалізації результатів тестування.

Сучасні методи аналізу результатів тестування можна умовно поділити на три основні групи: традиційні статистичні методи, інтелектуальні (аналітичні) підходи та системи комплексного моніторингу. Традиційні підходи ґрунтуються на використанні статистичних методів обробки результатів тестів, зокрема визначенні кількості успішних і

неуспішних тестів, середнього часу виконання, частоти помилок, рівня покриття коду тестами тощо. Такі методи є базовими і дозволяють отримати загальну оцінку якості, але не завжди забезпечують глибоке розуміння причин помилок чи взаємозв’язків між показниками.

Інтелектуальні підходи передбачають використання методів машинного навчання, аналізу даних та штучного інтелекту для виявлення закономірностей і прогнозування результатів майбутніх тестів. Використовуючи історичні дані про попередні тестування, можна навчити модель визначати ймовірність успішного проходження сертифікації або прогнозувати ризики появи критичних дефектів. Наприклад, класифікаційні моделі (Random Forest, Decision Tree, Logistic Regression) застосовуються для автоматичного розпізнавання типів помилок, а методи кластеризації — для групування результатів за схожими характеристиками. Такі інтелектуальні системи дозволяють не лише оцінювати стан продукту, а й приймати превентивні рішення ще до завершення циклу тестування.

Окремий напрям розвитку становлять системи комплексного моніторингу якості, які інтегруються з CI/CD-конвеєрами (рис. 1.2) (Jenkins, GitLab CI/CD, Azure DevOps) та системами управління тестуванням (TestRail, Zephyr, Allure). Вони забезпечують автоматичний збір, агрегацію та візуалізацію результатів тестування у вигляді інтерактивних дашбордів. Такі системи дозволяють відслідковувати тренди успішності тестів, навантаження на систему, стабільність релізів і швидко виявляти проблемні ділянки коду.

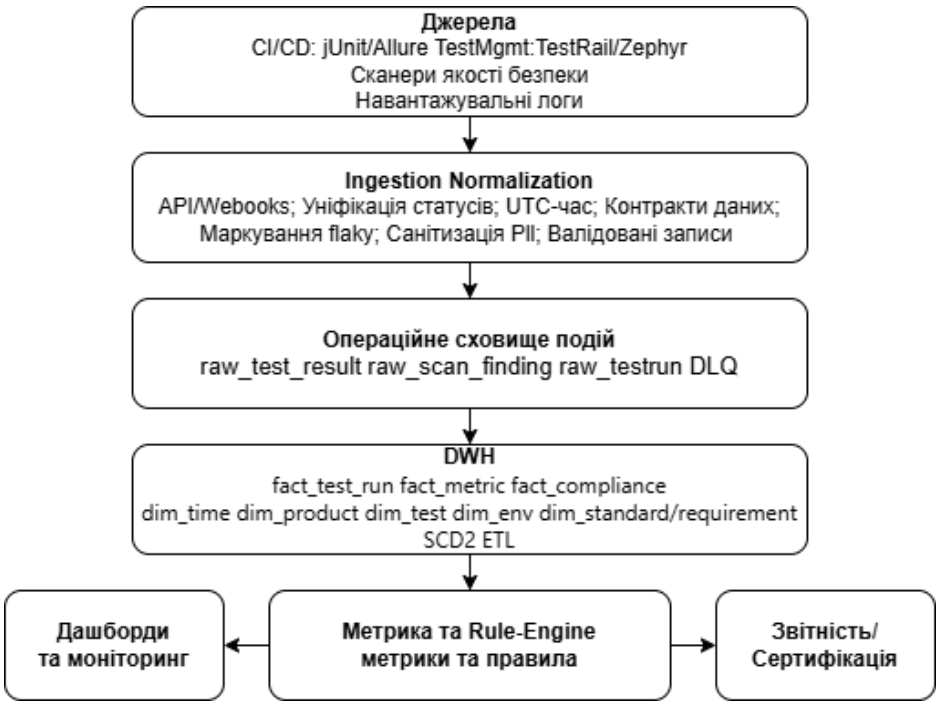

Рисунок 1.2 – Конвеєр аналізу результатів тестування

Зростає також роль метрик якості, які використовуються для кількісної оцінки результатів. Серед найбільш поширених — *Defect Density* (щільність дефектів), *Test Pass Rate* (рівень успішності тестів), *Mean Time To Failure* (середній час до відмови), *Code Coverage* (покриття коду тестами). Сукупність таких показників дозволяє формувати комплексну картину якості ПЗ та приймати обґрунтовані рішення щодо його готовності до сертифікації.

Сучасна тенденція розвитку аналізу результатів тестування полягає у переході від реактивного до прогнозного аналізу, коли система не лише відображає поточний стан, а й прогнозує можливі проблеми. Це особливо важливо у контексті сертифікації, де необхідно оцінювати відповідність продукту стандартам ще до завершення всіх етапів тестування.

Таким чином, сучасні підходи до аналізу результатів тестування ґрунтуються на поєднанні аналітичних інструментів, автоматизованих систем збору даних і методів інтелектуальної обробки інформації, що дозволяє підвищити точність оцінки якості, прискорити процес сертифікації та забезпечити прозорість у прийнятті рішень.

1.3 Проблеми та виклики сучасного тестування і сертифікації

Сучасні процеси тестування програмного забезпечення характеризуються різнорідністю джерел і форматів даних: результати автоматизованих тестів у CI/CD, протоколи систем керування тестуванням, звіти статичного аналізу, логи навантажувальних випробувань і безпекових сканерів. Відсутність єдиної моделі представлення та наскрізної трасованості «вимога → тест → дефект → реліз» призводить до фрагментації інформації та ускладнює об’єктивний висновок про якість продукту. Неоднорідність форматів (JUnit, Allure, CSV, JSON, ручні звіти), прогалини у метаданих (версія збірки, середовище виконання, конфігурація) та різний рівень довіри до джерел створюють бар’єри для порівнянності показників і коректної агрегації на рівні релізів.

Окремою проблемою є нестабільність тестів і «шум» у конвеєрах CI: flaky-тести, випадкові збої середовища, мережеві латентності або залежності від зовнішніх сервісів спотворюють індикатори якості та ускладнюють трендовий аналіз. Навіть високий відсоток покриття коду тестами не гарантує репрезентативності сценаріїв — у практиці часто бракує негативних, граничних і нефункціональних випробувань (продуктивність, стійкість, UX), а тестові набори з часом накопичують технічний борг: дублювання, застарілі перевірки, недостатню підтримку й ревізію.

Проблематика аналітики полягає не лише у збиранні чисел, а й у їх інтерпретації. Без стандартизованих визначень метрик (Pass Rate, Defect Density, SLA, MTTR/MTBF, Code Coverage) порівняння релізів або проєктів стає некоректним. Класичної описової статистики часто недостатньо для з’ясування причин дефектів, потрібні кореляційні та каузальні підходи, можливість «провалюватися» від агрегованих панелей до конкретних артефактів тесту, логу чи змін коду. Водночас зростає інтерес до прогнозного

аналізу та ML-моделей, які виявляють аномалії або оцінюють ризик провалу релізу; однак для їх прийнятності в аудиторському та сертифікаційному контексті необхідні пояснюваність, валідація, контроль версій і повторюваність результатів.

Сертифікація додає власний шар викликів. Формулювання вимог стандартів (ISO/IEC 25010, ISO/IEC 27001, галузеві норми) часто є багатозначними або мають різні рівні суворості, тож потрібен формальний мапінг метрик на критерії Pass/Fail і допустимі відхилення. Доказова база має бути повною та відтворюваною: результати тестів, протоколи, скріншоти, логи, артефакти мають зберігатися з гарантіями автентичності й цілісності, із журналами аудиту дій. Динаміка розвитку продукту ускладнює відповідність: кожна зміна версії ПЗ, вимоги чи тесту потребує трасованості, інакше попередні висновки втрачають валідність. Додаються вимоги конфіденційності та комплаєнсу (зокрема щодо персональних даних), які диктують контроль доступу, політики знеособлення та шифрування.

Не менш значущими є організаційні аспекти. Команди розробки, QA, SecOps, DevOps та зовнішні аудитори працюють у різних інструментах і часових горизонтах, що призводить до розривів у комунікації, дублювання або втрат інформації. Вартість і тривалість підготовки сертифікаційних звітів різко зростають, коли відсутня автоматизація формування вибірок і довідкових матеріалів. Залежність від окремих платформ тест-менеджменту чи сканерів створює вендор-локін і ускладнює уніфікацію метрик на рівні організації.

Масштаб і надійність інфраструктури аналітики стають критичними. Обсяги даних із тисяч тестів і гігабайтів логів потребують оптимізованих ETL-процесів, інкрементальних обчислень, розподілених сховищ і продуманого кешування. Для прийнятності в сертифікації необхідні механізми контролю цілісності (хеші, підписи), керування версіями артефактів і безперервне забезпечення відповідності (continuous compliance), коли політики стандартів

застосовуються під час кожного релізу, а не лише перед формальною перевіркою.

Сукупність окреслених викликів визначає вимоги до системи аналізу результатів: єдина модель даних і конектори до основних інструментів, процедури нормалізації та валідації якості, ідентифікація та ізоляція flaky-тестів, бібліотека стандартизованих метрик, rule-engine для формального мапінгу на стандарти, прозора аналітика з пояснюваними ML-моделями, журнали аудиту, контроль доступу, версіонування та автоматизована звітність. Лише поєднання цих принципів забезпечить об’єктивну, відтворювану та масштабовану сертифікацію програмного забезпечення.

1.4 Нормативні рамки та практики

Нормативне поле, у межах якого здійснюються тестування та сертифікація ПЗ, охоплює як моделі якості продукту, так і організаційні системи управління та галузеві регуляції. Для характеристики якості доцільно опиратися на ISO/IEC 25010, що визначає атрибути на кшталт надійності, продуктивності, безпеки, зручності, супроводжуваності й переносимості. Ці категорії транслюються у вимірювані метрики (Pass Rate, Coverage, P95/P99, Defect Density тощо) і стають основою для об’єктивної оцінки релізів. Процеси тестування описані в ISO/IEC 29119 (планування, проєктування, виконання, звітування) і добре узгоджуються з практиками CI/CD та V-моделлю розроблення.

Порівняння характеристик стандартів наведено в таблиці 1.1.

Таблиця 1.1

Стандарт	**Сфера**	**Ключовий фокус**	**Типові докази**	**Статус/оцінювання**

ISO/IEC 25010	Якість ПЗ	Характеристикиякості (напр. Reliability, Performance, Security)	Метрики Pass Rate, Coverage, P95/P99, Defect Density	Основа критеріївякості (без сертифікаціїпродукту)
ISO/IEC 27001	Інфобезпека(ISMS)	Організаційні й технічні контролі(Annex A)	Політики ISMS, журналидоступів/інцидентів, результати сканів	Сертифікаціяорганізації (аудит 3-стор.)
IEC 62304	ПЗ медичних виробів	ЖЦ ПЗ та трасованість для класів ризику A/B/C	Плани/Traceability, V&V протоколи, звіти про аномалії	Оцінкавідповідностіпроцесів/доказів
ISO/IEC 15408	Безпека IT-продуктів	SFR/SAR, рівнігарантій EAL1–EAL7	Security Target, незалежнетестування, звітивразливостей	Сертифікаціяпродукту (акредит. лабораторія)

Для інформаційної безпеки рамкою є ISO/IEC 27001 (ISMS): вона вимагає політик, управління ризиками, контролів доступу, журналювання та безперервного поліпшення. У контексті нашої системи технічні докази цієї відповідності походять із журналів аудиту, звітів сканерів уразливостей, конфігурацій шифрування та інцидент-менеджменту; вони агрегуються й відслідковуються поряд із тестовими метриками. Для **безпеки продуктів**застосовують також Common Criteria (ISO/IEC 15408) з рівнями гарантій EAL1–EAL7: вимоги безпеки (SFR) мають бути покриті тестами, а результати — відтворюваними та трасованими до артефактів. Практики OWASP (SAST/DAST, Top 10) доповнюють технічний пласт і дають стандартизовані профілі ризиків.

У галузях із підвищеними вимогами до безпеки й ризиків для життя (медичні вироби) використовують IEC 62304, що регламентує життєвий цикл ПЗ, класи ризику (A/B/C), повну трасованість «вимога → дизайн → тест» і протоколи V&V. Це прямо резонує з нашою моделлю даних: зв’язок Requirement–TestCase–TestResult і версійність артефактів (SCD2)

забезпечують доказовість на рівні, прийнятному для зовнішнього аудиту. У суміжних доменах аналогічну роль відіграють галузеві політики (наприклад, внутрішні SLA/SLO, вимоги регуляторів або замовників), які формалізуються як перевірні правила.

Запропонована система вбудовує ці рамки у rule-engine: вимога або контроль перетворюється на формулу виду «метрика–оператор–поріг–період–винятки», застосовується до агрегованих даних DWH і фіксується у протоколах відповідності з поясненням *measured vs threshold*. Завдяки трасованості доказів (посилання від правила до метрик, від метрик — до конкретних прогонів, логів, комітів і скан-звітів) досягається відтворюваність і прозорість висновків. У підсумку нормативні вимоги переходять із рівня «декларацій» у рівень continuous compliance: кожен реліз автоматично оцінюється за єдиними правилами, а експерти зосереджуються на перегляді політик, аналізі винятків і підвищенні зрілості процесів.

РОЗДІЛ 2

МЕТОДОЛОГІЯ АНАЛІЗУ РЕЗУЛЬТАТІВ ТЕТСУВАННЯ ТА ПІДТРИМКИ СЕРТИФІКАЦІЇ.

2.1 Модель даних і джерела

Методологія спирається на уніфіковану модель, що забезпечує наскрізну трасованість «вимога → тест-кейс → прогін → результат → дефект/знахідка → збірка → коміт → реліз → сертифікаційний висновок». Базові сутності охоплюють Requirement, TestCase, TestRun і TestResult, а також виробничий контекст Build і Commit (рис. 2.1). Для сертифікаційної придатності модель доповнюється Standard (класифікація вимог за нормативами), асоціативною таблицею між вимогами та тестами (Req_TestCase) для відображення зв’язків M:N, агрегаторами якості Metric і протоколом формальних перевірок Compliance_Check. За потреби додаються Product для ідентифікації версій і варіантів постачання, а також User/Audit_Log для фіксації дій та підтвердження автентичності артефактів.

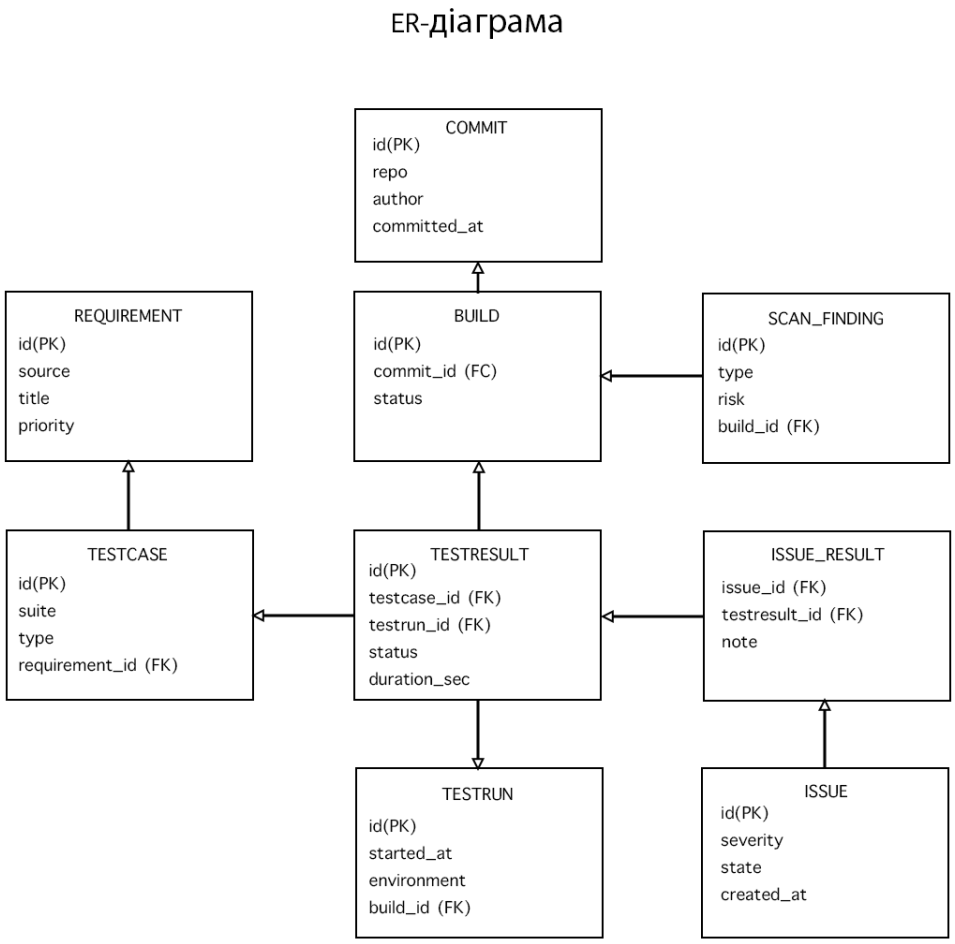

Рисунок 2.1 - ER-діаграма

Джерела даних включають автоматизовані прогони з CI/CD у форматах JUnit/Allure із артефактами виконання; каталоги тест-кейсів і їх метадані із систем керування тестуванням (на кшталт TestRail/Zephyr); результати статичного й динамічного аналізу якості та безпеки, зокрема вразливості, щільність дефектів і показники стилю; журнали навантажувальних випробувань із квантилями часу відгуку та пропускною здатністю; службові метадані релізів — середовище, конфігурації, фіче-флаги, посилання на коміти. Для кожного джерела встановлюється контракт даних із визначеними типами полів, часовими зонами, політикою оновлення та вимогами до мінімального набору атрибутів.

Ключова ідея уніфікації полягає в нормалізації статусів і часових полів, узгодженні ідентифікаторів та підвищенні якості метаданих. Статуси результатів приводяться до узгодженого словника («passed», «failed», «skipped», «xfailed»); часові мітки зберігаються в UTC; ідентифікатори уніфікуються до стабільних ключів. На етапі ingest/ETL виконується перевірка обов'язкових реквізитів — версії продукту, посилання на збірку, хеш коміту, середовище виконання — а також дедуплікація, відмітка нестабільних тестів, контроль цілісності артефактів за допомогою хеш-сум і, за потреби, знеособлення чутливих даних у логах.

Для аналітичних потреб застосовується DWH-схема типу «зірка». Факт fact_test_run синтезує дані TestResult із вимірами часу, продукту, тесту й середовища, зберігаючи тривалість, підсумковий статус та ознаки нестабільності. Факт fact_metric акумулює агреговані показники якості за періодом і версією — частку успішних тестів, квантилі часу відгуку, щільність дефектів, покриття тестами. Факт fact_compliance фіксує застосування політик відповідності: ідентифікатор вимоги та правила, виміряне значення, поріг, результат Pass/Fail і момент перевірки. Вимірювання dim_time, dim_product, dim_test, dim_env та dim_standard/requirement забезпечують швидкі зрізи за релізами, компонентами, середовищами й категоріями стандартів. Атрибути довідників ведуться як повільно змінні виміри з інтервалами дії, що зберігає історичну коректність звітів.

Мапінг «вимога ↔ тест» реалізується через Req_TestCase, оскільки один тест часто підтверджує кілька вимог, а одна вимога — кількома тестами різних рівнів. Саме на цьому зв'язку обчислюється покриття вимог, частка вимог із повністю успішними перевірками та виявляються «проблемні» вимоги зі сталими відмовами; ці показники прямо потрапляють у сертифікаційні розділи звіту. Паралельно агрегатор Metric підтримує різні області застосування — на рівні продукту, компонента або сервісу — з точною часовою прив'язкою для трендового аналізу.

Логічна модель ґрунтується на трасованості «вимога → тест-кейс → результат → метрика → правило → вердикт → звіт» і мінімальному, але обов'язковому контракті даних: build_id, commit_sha, environment, started_at_utc/ended_at_utc, testcase_id, уніфікований status, посилання на артефакти (логи, звіти сканера). Джерела включають CI/CD (JUnit/Allure), системи керування тестуванням (TestRail/Zephyr), сканери якості/безпеки (SAST/DAST), а також телеметрію продуктивності. На вході працюють нормалізація часових міток до UTC, дедуплікація і маркування flaky-тестів; усі сутності мають стабільні ключі (UUID/ULID) і зберігають походження (source) для аудиту. Така модель забезпечує однозначне з'єднання подій між системами та дозволяє відтворювати стан на момент релізу без втрати контексту.

Верифікація відповідності стандартам здійснюється rule-engine'ом, що інтерпретує політики у вигляді правил «метрика–оператор–поріг–період–винятки» і записує результати у Compliance_Check із поясненням «measured vs threshold». Це забезпечує відтворюваність висновків, прозорість логіки прийняття рішень і можливість аудиту. Додатково рольова модель і журнал дій гарантують контроль доступу, фіксацію затверджень і простежуваність, що є критично важливим для прийнятності сертифікаційних звітів.

Потік даних організований послідовно: надходження з CI/CD і тест-менеджменту, нормалізація та валідація, завантаження в операційне сховище «сирих» подій, інкрементальні трансформації до DWH-моделі, розрахунок метрик і виконання політик, побудова дашбордів та формування звіту. Разом така структурована модель поєднує детальний подієвий рівень, узагальнені індикатори якості та формальні протоколи відповідності, створюючи надійну основу для об'єктивного, повторюваного та аудиторсько придатного процесу сертифікації програмного забезпечення.

2.2 Метрики якості та відповідність

Система оцінювання спирається на узгоджений словник метрик, які кількісно відображають як функціональні, так і нефункціональні властивості програмного забезпечення, та на механізм формального зіставлення цих метрик із вимогами стандартів. Метрики визначаються на чітко окреслених рівнях деталізації — продукт, реліз, компонент, сервіс, середовище виконання — і в межах заданого часовго вікна спостереження. Для кожної метрики задаються одиниці виміру, спосіб агрегації, правила обробки пропусків та позаштатних значень, а також зв’язок із характеристиками якості (наприклад, продуктивність, надійність, безпека, сумісність, зручність) у термінах ISO/IEC 25010. Така семантична прив’язка дозволяє не лише вести технічний моніторинг, а й будувати аргументовані висновки для сертифікації.

Базові експлуатаційні індикатори включають частку успішних тестів і покриття тестами, квантилі часу відгуку (P95/P99) і частоту помилок, щільність дефектів та показники надійності, такі як середній час до відмови та середній час відновлення. Обчислення виконується над нормалізованими подіями тестування: успішність визначається як відношення кількості пройдених результатів до загальної кількості за обраний період; покриття розглядається у двох вимірах — кодове (лінії, гілки) і вимогове (частка вимог, що мають принаймні один прив’язаний і пройдений тест). Продуктивність описується розподілами латентності з акцентом на верхні квантилі, які краще відображають ризик деградацій; для шумних даних застосовується згладжування за ковзними вікнами та виняток очевидних артефактів. Показники безпеки підсумовуються через вагові профілі для різних рівнів критичності уразливостей, що дозволяє формувати інтегральний ризик безпеки релізу. Усі метрики зберігаються з атрибутами версійності, переліком джерел і позначками якості даних, щоб забезпечити відтворюваність розрахунків.

Поверх системи метрик працює правило-орієнтований механізм відповідності. Кожне правило визначається як п’ятірка «метрика – оператор – поріг – період – винятки» і має детермінований результат перевірки з поясненням. Наприклад, “PassRate ≥ 95% за 14 днів для релізної гілки” або “P95Latency ≤ 2,0 с для сервісу авторизації в середовищі production”, або “Кількість критичних уразливостей = 0 на збірку”. Підтримуються як прості порогові оператори, так і складені вирази з вагами та логічними зв’язками для формування композитних умов. Для кожного правила фіксуються версія, дата набуття чинності, власник і посилання на відповідну норму стандарту; це дозволяє проводити контроль змін і ретроспективний аналіз. Результати застосування правил записуються у протокол перевірок із вказанням виміряного значення, порога, контексту (версія продукту, середовище) і відбитку часу виконання; у разі невідповідності додаються автоматично згенеровані рекомендації та посилання на артефакти, що підтверджують висновок.

Аналітичний шар реалізовано як «зірку» з фактами fact_test_run, fact_metric, fact_compliance та вимірами dim_time, dim_product, dim_test, dim_env, dim_standard/requirement. Виміри ведуться як SCD2 з valid_from/valid_to, що гарантує історичну коректність: кожне значення метрики та кожен протокол відповідності інтерпретуються в контексті тодішніх версій продукту, правил і визначень. Інкрементальні ETL-джоби переносять події з операційного шару, обчислюють агрегати (Pass Rate, Coverage, P95/P99, Defect Density, MTTR) і зшивають їх із вимірами; великі таблиці партиціюються за часом, для запитів використовуються індекси та BRIN/bitmap, що дає змогу одночасно підтримувати оперативну аналітику і швидку генерацію звітів.

Унікальною для сертифікаційних сценаріїв є підтримка винятків і делегування рішень. Для правил можуть задаватися дозволені відхилення у визначених межах часу або обсягу — наприклад, короткочасні піки

латентності під час розгортання — із вимогою компенсації в наступні періоди. Якщо виняток ініціює експерт, система фіксує підстави, термін дії та контрзаходи, що унеможливлює «мовчазне» заниження стандартів. Конфлікти правил розв’язуються через пріоритизацію ієрархій: жорсткі норми безпеки мають пріоритет над зручністю чи продуктивністю, а глобальні вимоги стандарту — над локальними політиками компонентів; ця логіка формально описується у каталозі політик і застосовується послідовно.

Для підвищення інформативності підсумкова відповідність може обчислюватися як інтегральний індекс, що поєднує результати кількох правил за вагами, узгодженими з ризик-моделлю організації. Індекс інтерпретується у триярусній шкалі “зелений/жовтий/червоний”, а разом із ним подається оцінка невизначеності — довірчі межі для метрик, чутливість індексу до ключових параметрів та перелік найвпливовіших правил. Там, де використовується прогнозна аналітика, метрики калібруються, імовірнісні рішення супроводжуються оцінкою якості передбачень і коротким поясненням внеску факторів, що робить висновок зрозумілим для аудиту.

Завершує методичний контур життєвий цикл метрик і правил. Нові метрики проходять стадію експериментальної експлуатації з паралельним обліком і валідацією; після затвердження отримують стабільні ідентифікатори та входять до звітності. Зміни у формулах, агрегуванні або порогах відстежуються через версії з чіткими датами набуття чинності; у звітах за історичні періоди завжди відтворюються саме ті визначення, що діяли на момент виміру. Така дисципліна визначень, разом із прозорою реалізацією rule-engine, гарантує репродукованість, пояснюваність і прийнятність результатів для сертифікаційних органів, водночас зберігаючи гнучкість для різних доменів і змін нормативної бази.

2.3 Аналітичні та інтелектуальні методи

Аналітичний шар поєднує описову статистику, часові ряди та причинно-наслідковий аналіз, перетворюючи сирі події тестування на індикатори, придатні для рішень щодо сертифікації. Базовий рівень охоплює оцінку розподілів тривалостей тестів і латентності сервісів, довірчі інтервали для частки успішних запусків і похибок вимірювань, а також стабільні тренди із сезонністю та точками змін. Для відокремлення випадкових флуктуацій від системних деградацій застосовуються ковзні вікна, згладжування (EMA), декомпозиція STL і детектори змін, що дає змогу коректно інтерпретувати короткочасні сплески, спричинені релізами або інфраструктурними роботами. Якість коду та процесу оцінюється через узгоджені метрики надійності (MTBF/MTTR), щільності дефектів, покриття й середні часи виправлення, а їх варіативність аналізується бутстрепом, щоб уникати хибних висновків на малих вибірках.

Прогнозний аналіз використовує моделі машинного навчання для оцінки ймовірності успішного проходження релізом порогів відповідності або появи критичних дефектів у найближчому циклі. Класифікаційні підходи (логістична регресія, дерева рішень, ансамблі на кшталт Random Forest/Gradient Boosting) навчаються на історіях релізів із ознаками, збагаченими контекстом: частки невдалих тестів по компонентах, аномалії латентності P95/P99, інтенсивність дефектів за категоріями, частота “flaky” та повторних прогонів, розмір змін у коді, рух у гілках, конфігурації середовищ. Для часових завдань застосовуються моделі прогнозування рядів (ARIMA/Prophet/ETS) або рекурентні варіанти, коли важлива динаміка показників. Регресійні моделі оцінюють очікувану латентність або навантажувальну пропускну здатність за умовою конкретної конфігурації, що допомагає робити превентивні рішення до запуску повного набору тестів.

Надійність рішень забезпечується правильною валідацією: розділенням на тренувальний і відкладений періоди, k-fold із часовим порядком, перевіркою на переносимість між продуктами й середовищами. Для

класифікації застосовуються ROC-AUC/PR-AUC, F1 і матриці помилок у розрізі ризик-категорій, а також калібрування ймовірностей (Platt/Isotonic), що дає змогу узгодити пороги rule-engine з реальними ризиками. Ізраноцінені дані компенсуються стратифікованими вибірками, вагами класів або методами балансування; особлива увага приділяється уникненню витоку даних, коли інформація з майбутнього випадково потрапляє в ознаки. Для обмеження переобучення застосовуються регуляризація, скоринг важливостей і відбір ознак, а стабільність показників контролюється періодичним переонавчанням і моніторингом дрейфу.

Пояснюваність моделей є критичною для сертифікації, тому разом із прогнозом подається локальне та глобальне тлумачення впливу ознак. Методи на кшталт SHAP забезпечують прозору декомпозицію внесків: які метрики (наприклад, частка failed у критичному модулі або зсув P95) найбільше знижують імовірність відповідності. Для прийнятності в аудиті доповнюється контрфактичним аналізом: які мінімальні зміни у процесі (зменшення повторних прогонів, підвищення покриття вимог) переведуть прогноз у зону “Pass”. Там, де це доречно, застосовуються байєсівські підходи з апостеріорними розподілами параметрів і інтервалами невизначеності, що дає змогу приймати обачні рішення за умов нестачі даних.

Виявлення аномалій працює у двох площинах: точкові викиди у потоках метрик (латентність, помилки, часи тестів) і контекстні відхилення у патернах прогонів. Для першого підходять ізоляційні ліси чи одномодові SVM, для другого — моделі реконструкції (автоенкодери) або щільнісні методи, що відстежують рідкісні комбінації конфігурацій і змін коду. Важливо поєднати аномалії з причинним шаром: якщо відхилення латентності співпадає з релізом конкретного компонента, система додає посилання на відповідний коміт і набір тестів, які найбільше змінилися, що суттєво скорочує час реагування.

Нерідко значущим джерелом інформації є текстові артефакти — логи, повідомлення про помилки, описи дефектів. Легковагові NLP-методи виконують нормалізацію й класифікацію типів збоїв, групування схожих інцидентів і виділення корисних сутностей (ідентифікатори тестів, компонентів, винятки). Це дозволяє будувати “словники симптомів”, які в поєднанні з кількісними метриками підсилюють як виявлення рецидивів, так і формування рекомендацій.

Інтеграція з rule-engine забезпечує перехід від імовірнісних оцінок до формальних рішень. Прогнози та аномалії перетворюються на сигнали, що ініціюють перевірку правил або запуск компенсуючих процедур; невизначеність моделі відображається у поліціях як “жовта зона” з вимогою додаткової перевірки. Кожне автоматизоване рішення супроводжується протоколом із посиланнями на вихідні артефакти, версіями даних і моделей, що уможливлює аудит і повторюваність. Поверх цього діє контур MLOps/Model Governance: контроль версій фіч і моделей, відслідковування дрейфу даних, алерти на деградацію якості та регулярні переоцінки адекватності порогів у правилах.

Завдяки такому поєднанню статистики, прогнозного моделювання, виявлення аномалій, пояснюваності та керування життєвим циклом аналітика стає не просто інструментом моніторингу, а й механізмом забезпечення обґрунтованих, відтворюваних і прийнятних для сертифікації рішень. Це зводить ризик суб’єктивності до мінімуму, робить процес прозорим для експертів і дозволяє масштабувати практику continuous compliance між командами та продуктами.

2.4 Візуалізація, звітність і дизайн експерименту

Візуалізаційний шар перетворює показники якості та результати перевірок на узгоджену систему панелей, що підтримує як оперативний моніторинг, так і сертифікаційні рішення. Базою слугує головна панель відповідності стандартам, де у єдиному полі представлені статуси правил із rule-engine у розрізі продукту, версії, середовища та часових вікон. Кожен індикатор має чітку семантику: зелений означає стабільне виконання порога, жовтий сигналізує про тимчасові відхилення в межах допустимих винятків, червоний фіксує невідповідність із негайною потребою втручання. Натискання на індикатор відкриває контекст: виміряні значення, порогові умови, артефакти тестів, логи прогонів, посилання на вимогу стандарту та історію змін, що забезпечує наскрізну трасованість від агрегату до першоджерела.

Поруч із панеллю відповідності розміщуються трендові уявлення ключових метрик: частка успішних тестів, квантилі часу відгуку, щільність дефектів, показники безпеки. Візуалізації акцентують на верхніх квантилях і сезонності, дозволяючи бачити як повільні дрейфи, так і короткочасні піки, прив’язані до релізів чи змін конфігурацій. Теплові карти доповнюють огляд, відображаючи просторовий розподіл дефектів і невідповідностей по компонентах, сервісах та середовищах, а також швидко ідентифікуючи «гарячі зони». Усі графіки підтримують фільтри за продуктом, версією, середовищем, категорією вимог і часовим діапазоном, а також мають режим «drill-down» до рівня конкретного тест-кейсу, прогону або коміту.

Механізми сповіщення інтегруються з візуалізаціями й політиками. Для критичних правил налаштовуються канали повідомлень із детальним контекстом: вимога, виміряне значення, очікуваний поріг, посилання на артефакти та рекомендовані дії. Сповіщення групуються та придушуються в разі повторюваних ідентичних подій, щоб уникати «шуму». Для жовтої зони використовуються нагадування з відкладеною ескалацією, якщо показники не повертаються до норми в оголошений період.

Звітність реалізується як керований конвеєр формування документів для експертів і сертифікаційних органів. Шаблон звіту має стабільну структуру: резюме з підсумковим вердиктом, опис методики збору та нормалізації даних, таблиці метрик із довірчими межами, протоколи перевірок правил із поясненнями «measured vs threshold», розділ відповідності вимогам із трасованістю «вимога ↔ тест», додатки з артефактами (фрагменти логів, скріни, витяги з систем керування тестами). Кожне число супроводжується вказівкою версійності: версія продукту, коміту, тест-кейсу, правила, а також ідентифікатором збірки звіту, що гарантує відтворюваність. Підтримуються два режими: оперативний (інкрементальні звіти на реліз/спринт) і формальний (пакет для сертифікації з фіксацією стану на дату). Для прозорості передбачено журнал підписів: хто переглянув, хто візував і хто затвердив документ.

Дизайн експерименту визначає, як саме підтверджується ефективність методології та інструментарію. Емпірична база формується з історичних прогонів тестів і релізів за достатній період, щоб охопити різні профілі навантаження й сезонність. Порівняння проводиться між базовою практикою (ручна агрегація й точкові метрики без формальних правил) і запропонованим підходом (уніфікована модель даних, rule-engine, дашборди, автоматизована звітність). Для кожного релізу попередньо фіксуються незалежні еталонні оцінки якості від експертів, що служать «золотим стандартом» для валідації. Критеріями є точність і повнота виявлення невідповідностей, стабільність рішень між релізами, зменшення часу на підготовку висновку та обсяг ручних дій. Якщо застосовуються прогностичні моделі, додаються якість калібрування ймовірностей, ROC-AUC/PR-AUC і стійкість прогнозів на відкладених періодах.

Щоб уникнути систематичних помилок, експериментальна процедура враховує розділення даних у часі, виключення витоку інформації між тренувальними й тестовими наборами та стратифікацію за типами

компонентів і середовищ. У протоколі відтворюваності описуються версії джерел, схеми перетворень, пороги правил і параметри візуалізацій; додатково фіксується, які саме рішення були прийняті на основі панелей і звітів та який мали практичний ефект (скорочення перезапусків, зменшення інцидентів у продукшні, швидше закриття дефектів). Після основного циклу оцінювання проводиться чутливісний аналіз: як змінюються висновки за варіації порогів, вікон згладжування, правил агрегації та визначень метрик. Це дозволяє сформувати рекомендації щодо робочих налаштувань системи та окреслити межі її застосовності.

У підсумку візуалізація, звітність і експериментальний дизайн утворюють замкнений контур: дашборди забезпечують прозорість і оперативність, звіти — формальну прийнятність і відтворюваність, а експеримент — доказову базу ефективності методики. Така інтеграція мінімізує суб’єктивність, підвищує довіру з боку сертифікаційних органів і створює основу для практики безперервної відповідності, коли кожен реліз оцінюється за єдиними правилами, з однаковою точністю та пояснюваністю.

РОЗДІЛ 3

РЕАЛІЗАЦІЯ ТА ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ

3.1. Архітектура рішення

Архітектура будується як конвеєр із чітко розмежованими шарами: ingestion та нормалізація даних → операційне сховище “сирих” подій → DWH з моделлю «зірка» → аналітика й rule-engine → візуалізація та звітність. Така стратифікація мінімізує зв’язність між компонентами, спрощує еволюцію системи та забезпечує відтворюваність сертифікаційних висновків. Логічна модель безпосередньо спирається на ER-схему (сутності Requirement, TestCase, TestRun/TestResult, Build/Commit, Issue/Issue_Result, Metric, Compliance_Check, Standard, Product), а також на архітектурну схему (рис.3.1).

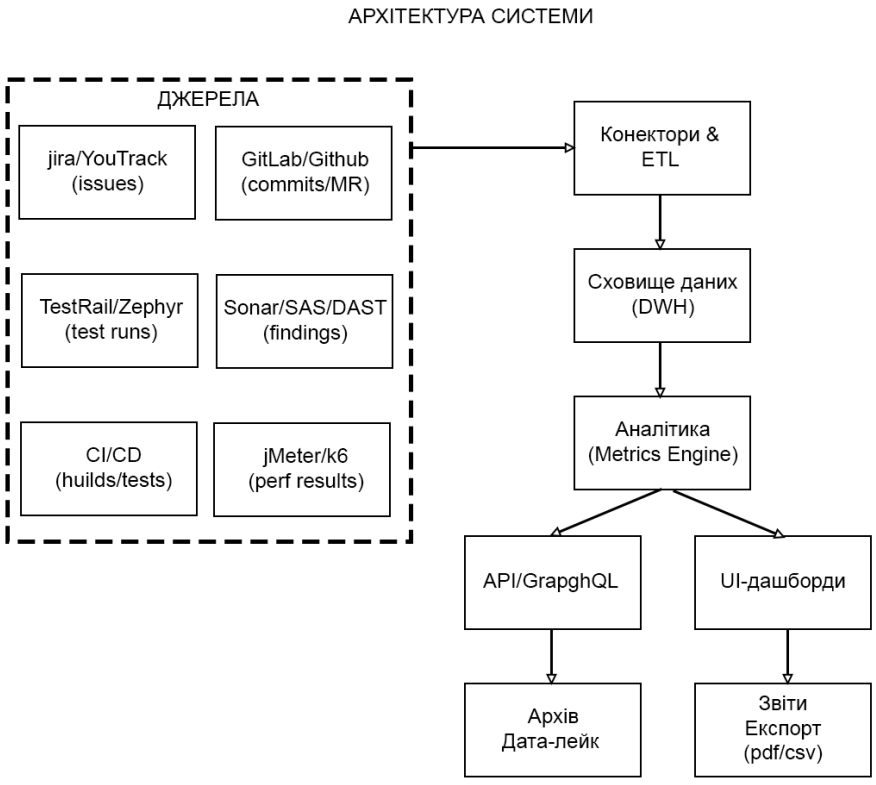

Рисунок 3.1 - Архітектура системи

Шар ingestion і нормалізації приймає події з CI/CD (JUnit/Allure), систем керування тестуванням (TestRail/Zephyr), сканерів якості/безпеки та журналів навантаження. На вході діє контракт даних з мінімально необхідними полями: build_id, commit_sha, environment, started_at/finished_at, testcase_id, status ∈ {passed, failed, skipped, xfailed}, для сканерів — finding_id, severity, location, scanner. Усі часові мітки переводяться в UTC, статуси приводяться до єдиного словника, ідентифікатори уніфікуються (UUID/ULID). Валідація контролює наявність ключових метаданих, а також маркує flaky-тести (евристики: нестабільні результати в межах короткого вікна, залежність від середовища). На цьому ж етапі виконується санітизація логів (знеособлення потенційно чутливих фрагментів) і розрахунок контрольних сум артефактів (хеші) для подальшого аудиту.

Операційне сховище подій (OLTP-рівень) зберігає нормалізовані записи протягом короткого періоду для повторного прогону конвеєрів у разі збоїв. Модель максимально близька до “подієвої”: таблиці на кшталт raw_test_result, raw_scan_finding, raw_testrun. Тут важливі ідемпотентність і детермінованістьзавантажень: повторне надходження того самого прогону має безпечно оновлювати або ігнорувати запис (ключі на основі build_id + testcase_id + started_at). Усі зміни журналюються: хто, коли і що саме перезавантажував, із посиланням на оригінальні джерела.

DWH-шар із моделлю «зірка» виконує інкрементальний ETL з операційного сховища до агрегованої структури: факти fact_test_run (результати тестів у розрізі часу/продукту/тесту/середовища), fact_metric(агреговані показники: Pass Rate, Coverage, P95/P99, Defect Density, MTTR/MTBF, Error Rate, кількість критичних уразливостей), fact_compliance (протоколи застосування правил відповідності), а також виміри dim_time,

dim_product, dim_test, dim_env, dim_standard/dim_requirement. Довідники ведуться як повільно змінні виміри (SCD Type 2) з полями valid_from/valid_to, що гарантує історичну коректність звітів: кожен сертифікаційний висновок відтворюється в контексті саме тих визначень і правил, які діяли на момент перевірки.

Аналітичний шар і rule-engine життєво важливі для сертифікації. Аналітичні джоби обчислюють тренди та розподіли (ковзні вікна, квантилі латентності, стабільність метрик), виділяють аномалії (раптові піки, деградації), а також агрегують requirements coverage через зв'язку Req_TestCase. Rule-engine зчитує каталог політик (YAML/JSON) з версійністю і датами чинності та трансформує їх у формальні перевірки виду «метрика – оператор – поріг – період – винятки». Наприклад: *"PassRate ≥ 95% протягом останніх 14 днів у гілці release"*, *"P95 latency ≤ 2.0 с у prod за 7 днів"*, *"Critical vulns = 0 на кожну збірку"*. Результат кожної перевірки записується до fact_compliance у форматі measured vs threshold з контекстом (продукт/версія/середовище/час), що дає детермінований слід для аудиту.

Візуалізація та звітність базуються на готових агрегатах DWH і протоколах rule-engine. Головна панель показує статус відповідності у трьох кольорах (зелений/жовтий/червоний) з можливістю drill-down до первинних артефактів: прогонів, логів, комітів. Окремі панелі відображають тренди Pass Rate/Latency/Defects, теплові карти «гарячих» компонентів і таблиці відповідності вимогам. Конвеєр звітності формує оперативні (на реліз/спринт) та формальні сертифікаційні документи (PDF/Word) з резюме, описом методології, таблицями метрик із довірчими межами, протоколами перевірок і додатками (виписки з систем тест-менеджменту, фрагменти логів, скріншоти). Кожне число у звіті має атрибути версійності (версія продукту, коміту, тест-кейсу, правила), що гарантує повторюваність.

Доступ, безпека та аудит реалізовано через рольову модель (QA/Dev/Manager/Expert) і централізований AUDIT_LOG: створення/зміна правил, перезавантаження даних, експорт звітів, акти затвердження. Артефакти й таблиці можуть підписуватися контрольними сумами; конфіденційні поля логів — знеособлюватися або шифруватися. Це критично для прийнятності результатів у сертифікаційних органах та відповідності регуляціям.

Експлуатаційні властивості забезпечуються за рахунок масштабованого розгортання і спостережуваності. Сервіси контейнеризуються (Docker) і оркеструються у Kubernetes або через Docker-Compose для навчально-дослідного стенда. Observability включає метрики самих конвеєрів (черги подій, тривалість ETL, стан джоб), технічні логи й алерти (затримки, помилки парсерів, невідповідність контракту даних). Підтримуються ретраї з back-off, черги на dead-letter для проблемних подій і механізми ідемпотентного перезавантаження. Бекапи DWH і журналів аудиту — за розкладом; для критичних компонентів передбачене горизонтальне масштабування.

Типовий сценарій проходження даних виглядає так. Реліз у CI запускає тести та сканери → артефакти JUnit/Allure і звіти сканерів потрапляють до ingestion через API/webhook → нормалізація перетворює їх у подієві записи та кладе в операційне сховище → інкрементальний ETL завантажує події до fact_test_run, перераховує fact_metric, оцінює правила й записує fact_compliance → дашборди оновлюються, а у разі відхилень надсилаються сповіщення → за потреби формується PDF/DOCX звіт із підсумковим вердиктом (Pass/Fail/Conditional) і трасованістю до першоджерел.

Нефункціональні вимоги зосереджені на відтворюваності, прозорості та переносимості. Всі трансформації версіонуються, параметри правил мають чіткі дати чинності, а визначення метрик — фіксовані формули із зазначенням рівня застосування (продукт/компонент/середовище) та вікна спостереження.

Інтеграції з інструментами виконані через тонкі конектори; у разі їх заміни (інший тест-менеджер або CI) достатньо адаптувати ingestion без змін у DWH і вище. Саме це дозволяє масштабувати підхід на нові продукти й домени та підтримувати continuous compliance — коли кожен реліз автоматично оцінюється за єдиними, прозорими й аудиторсько придатними правилами.

3.2. Реалізація ключових модулів

Реалізація побудована як набір незалежних сервісів із чіткими контрактами даних і версіонуванням трансформацій. Кожен модуль має власні тести, журнали подій та показники спостережуваності, а взаємодія відбувається через події й REST/API-виклики. Нижче — практичні деталі, достатні, щоб відтворити систему на лабораторному або пілотному стенді.

Джерела інтегруються адаптерами:

- CI/CD тести: прийом JUnit/Allure звітів через webhook або з артефактів збірки. Парсер уніфікує статуси до passed/failed/skipped/xfailed, витягає тривалість, шлях тесту, мітки компонентів.
- Test Management (TestRail/Zephyr): періодичний імпорт каталогу тест-кейсів та їх зв’язків із вимогами. Під час імпорту застосовується дедуплікація за external_id.
- Сканери якості/безпеки: імпорт JSON/CSV зі списками знахідок (finding_id, severity, location, scanner, build_id).
- Навантажувальні логи/метрики: парсинг латентності/помилок (наприклад, з експорту з Prometheus/InfluxDB) до уніфікованого формату.

Контракт події після нормалізації відображено на рисунку 3.1.

Рисунок 3.1 - контракт події після нормалізації

Ідемпотентність забезпечується унікальним ключем події (source, build_id, testcase_id, started_at_utc). Повторні надходження призводять до upsert. Критичні поля перевіряються до збереження; події з неповними метаданими відправляються до dead-letter queue із причиною відхилення. Весь інлет приводить часові поля до UTC; логи санітизуються від PII за допомогою шаблонів/регулярних виразів.

Нормалізовані записи потрапляють у «сирі» таблиці:

- raw_test_result(source, build_id, testcase_id, started_at_utc, status, duration_ms, labels, hash)
- raw_scan_finding(build_id, finding_id, severity, location, scanner)
- raw_testrun(build_id, environment, started_at_utc, ended_at_utc)

Далі інкрементальний ETL переносить дані у DWH:

- Злиття raw_* у факти та виміри; створення відсутніх записів у dim_test, dim_product, dim_env.
- Маркування flaky за евристикою: один і той самий testcase_id у межах 24 годин має різні статуси при незмінному environment.

- Розрахунок метрик у робочих таблицях (stage) і комміт у fact_metric батчами.

Приклад обчислення Pass Rate у DWH відображено на рисунку 3.2.

```
INSERT INTO fact_metric (product_id, metric_name, value, collected_at, scope)
SELECT p.id, 'pass_rate', 100.0*sum(CASE WHEN r.status='passed' THEN 1 ELSE 0 END)/cou
       date_trunc('day', r.started_at_utc), 'product:release'
FROM fact_test_run r
JOIN dim_product p ON p.build_id = r.build_id
WHERE r.started_at_utc >= now() - interval '14 day'
GROUP BY p.id, date_trunc('day', r.started_at_utc);
```

Рисунок 3.2 - Pass Rate у DWH

Аналітика працює поверх фактів:

- Ковзні вікна для латентності (P95/P99) й помилок; відсікання артефактів за IQR/правилом 3σ.
- Покриття вимог: з’єднання REQ_TESTCASE з останніми результатами тестів; статус вимоги — PASS, якщо всі пов’язані тести пройдені у вибраному вікні.
- Надійність процесу: MTTR/MTBF за подіями інцидентів/дефектів.

Всі формули фіксуються у «каталозі метрик» із версіями: id, name, formula_ref, scope, window, owner, valid_from/valid_to. Це дозволяє відтворити історичні значення навіть після зміни визначень.

Правила зберігаються у конфігураціях (YAML/JSON) та інтерпретуються виконуючим модулем.

Приклад каталогу зображено на рисунку 3.3.

```yaml
rules:
  - id: R-01
    name: ISO25010.Reliability.PassRate
    metric: pass_rate
    operator: ">="
    threshold: 95.0
    window: "14d"
    scope: "product:release"
    exceptions:
      - type: flaky_allowance
        value: 2   # до 2 перезапусків/добу
    valid_from: "2025-01-01"

  - id: R-02
    name: Perf.SLA.P95
    metric: latency_p95
    operator: "<="
    threshold: 2.0   # seconds
    window: "7d"
    scope: "service:auth@prod"
```

Рисунок 3.3 - Приклад каталогу метрик

Виконання відбувається циклічно або подійно (on-build). Engine підтягує з DWH релевантні значення метрик у потрібному вікні, оцінює умови, застосовує винятки (наприклад, ігнорує піки під час деплою до 30 хв), і записує протокол у fact_compliance: *(product_id, requirement_id?, rule_id, measured, threshold, pass_fail, checked_at, context)*

Для FAIL/WARN додається коротка рекомендація (rule hint): “підвищити coverage до ≥90%”, “зменшити P95 < 2.0 с; див. сервіс auth”.

Візуалізація читає вже готові агрегати:

- Панель відповідності: мапа правил × продукти/релізи зі статусами, фільтрами за середовищем і drill-down до fact_compliance.

- Тренди: Pass Rate, P95/P99, Defect Density; анотації релізів/деплоїв.
- Покриття вимог: таблиця/теплова карта з розрізом за категоріями стандартів.

Конвеєр звітів збирає:

1. резюме й вердикт (Pass/Fail/Conditional),
2. методику та джерела,
3. таблиці метрик із довірчими межами,
4. протоколи перевірок правил (measured vs threshold),
5. покриття вимог і посилання на артефакти.

Кожний аркуш має атрибути версійності: product_version, commit_sha, ruleset_version, metric_catalog_version. Експорт у PDF/DOCX здійснюється зі стандартизованого шаблону (див. згенерований шаблон у попередньому кроці).

Надійність, продуктивність і спостережуваність визначається:

- Idempotency & retries: усі операції з базою — через UPSERT/MERGE; повторні спроби з back-off; проблемні події — у DLQ з причиною.
- Performance: індекси на (build_id), (testrun_id, testcase_id), (requirement_id), партиціювання фактів за датою; інкрементальні підрахунки метрик, кеш квантилів.
- Observability: техметрики конвеєра (час ETL, довжина черги, rate помилок), журнали з кореляційними ID (build_id, rule_id), алерти на SLA дашбордів і експорту звітів.

- Безпека й аудит: ролі (QA/Dev/Manager/Expert), AUDIT_LOG для змін правил і експорту документів; хеші артефактів у звітних додатках.

Тестування і верифікація модулів проходитимуть за принципами:

- Юніт-тести парсерів форматів (JUnit/Allure/сканери) з edge-кейсами.
- Контракт-тести джерел (зміна схеми не повинна ламати ingestion).
- Інтеграційні тести ETL на синтетичних вибірках із відомим еталоном метрик.
- Golden tests rule-engine: набір історичних сесій з фіксованими очікуваними вердиктами.
- Валідація звітів: перевірка відповідності номерів версій і посилань на артефакти, цілісність PDF/DOCX.

Після реалізації цих компонентів система відтворювано перетворює “сирі” результати тестів та сканів на узгоджені метрики, формально оцінює відповідність політикам і випускає сертифікаційні звіти з повною трасованістю до першоджерел. Якщо потрібно, можу додати ескіз SQL для всіх таблиць або YAML-шаблон каталогу правил під конкретний стандарт (з твоїми порогами).

3.3. Дизайн експерименту

Метою експерименту є доказова оцінка того, що запропонована архітектура (уніфікована модель даних + rule-engine + DWH «зірка» + дашборди/звітність) підвищує якість виявлення невідповідностей і скорочує час підготовки сертифікаційного висновку порівняно з базовою практикою (ручні підрахунки, неуніфіковані метрики, без формальних правил).

Припущено такі гіпотези:

- точність і повнота виявлення невідповідностей (відносно еталонних експертних оцінок) у запропонованому підході вищі або не нижчі за baseline.

- середній час підготовки сертифікаційного висновку скорочується щонайменше на 30%.
- стабільність вердиктів між релізами (міжрелізна варіація) зростає, а частка «хибних тривог» зменшується.
- (за наявності прогнозних моделей): калібрування імовірностей адекватне (Brier score, reliability plot), ROC-AUC/PR-AUC ≥ наперед узгоджених порогів.

Взято 1–2 продукти з різними компонентами/сервісами, 3–6 місяців історії релізів, повні артефакти тестування (JUnit/Allure), керування тестами (TestRail/Zephyr), сканери якості/безпеки й навантажувальні метрики. Кожен реліз має прив'язку до build_id/commit_sha і середовища (dev/test/stage/prod).

Незалежна змінна — методика (Baseline vs Proposed). Залежні: (1) Precision/Recall/F1 виявлення невідповідностей (по вимогах/правилах), (2) T_prepare — час підготовки висновку (збір даних → підпис), (3) Stability — варіація вердиктів між релізами, (4) False Alarm Rate — частка кейсів, де Baseline/Proposed помилково фіксують невідповідність, (5) за ML — ROC-AUC/PR-AUC, Brier score, калібрування.

Для підмножини релізів (не менше 20–30) готуються незалежні експертні висновки: список невідповідностей з посиланнями на артефакти, узгоджений за процедурою «двох рецензентів + арбітр». Саме з ним порівнюються Baseline і Proposed.

Процедура:

1. Відтворення Baseline: повторити звичний спосіб команди (ручні таблиці/Excel, локальні скрипти без формального rule-engine). Зафіксувати витрати часу (start/stop тайм-лог), отримати фінальний список невідповідностей і вердикт.

2. Запуск Proposed: проганяється конвеєр ingestion → DWH → розрахунок метрик → rule-engine → дашборди/звіт/вердикт. Логується фактичний час, збираються протоколи fact_compliance, «measured vs threshold».
3. Порівняння з еталоном: для кожного релізу побудувати матрицю помилок (TP/FP/FN/TN) щодо еталону, порахувати Precision/Recall/F1, FAR.
4. Міжрелізна стабільність: для однакових правил обчислити дисперсію/коеф. варіації Pass/Fail між послідовними релізами; оцінити зниження «гойдалок» у Proposed.
5. Аналіз часу: знеособити ролі, порівняти розподіли T_prepare між Baseline і Proposed; за необхідності — непараметричний тест (Mann-Whitney) або парний тест (якщо релізи співставні).
6. ML-додаток (опційно): якщо є прогнозні моделі — навчання на історії N-1 місяців, тест на останньому місяці; метрики ROC-AUC/PR-AUC, калібрування (isotonic/Platt), reliability plot.
7. Чутливісний аналіз: варіювати пороги правил (±5–10%), розміри ковзних вікон (7/14/28 днів), фільтри «flaky», і перевірити стабільність висновків.
8. Документування відтворюваності: версії джерел, Git-хеші ETL і rule-catalog, ідентифікатори даних у DWH, список змін під час експерименту.

Агрегати будуються на DWH-фактах: fact_test_run → Pass Rate/статуси/flaky; fact_metric → P95/P99, Defect Density, Coverage; fact_compliance → підсумок правил. Для шумних рядів застосовуються ковзні вікна/стійкі оцінки (медіана, IQR-фільтр), для подій релізів — анотації на графіках. Довірчі інтервали для пропорцій — Wilson/Jeffreys; порівняння середніх часів — t-тест або непараметричний еквівалент.

Проєкт вважається успішним, якщо: (i) F1(Proposed) ≥ F1(Baseline) і не нижче еталонного порогу (наприклад, 0.9); (ii) середнє T_prepare знижено ≥ 30% (95% CI); (iii) False Alarm Rate не зростає і, бажано, зменшується; (iv) у

звітності наявні відтворювані посилання на артефакти, а rule-engine забезпечує прозору логіку рішень. Для ML — ROC-AUC ≥ 0.8, прийнятне калібрування (Brier ≤ 0.2) або чітка користь у пріоритизації перевірок.

Внутрішня: витік даних між train/test — розділення у часі, lock-файли версій. Зовнішня: специфіка продукту — включити два різні продукти або різні середовища. Конфундери: оновлення інфраструктури/правил — фіксувати версії, повторити експеримент після змін. Людський фактор у Baseline — стандартизований чек-лист, подвійна перевірка тайм-логів.

Артефакти проходять знеособлення; доступ до raw_* обмежений; у публічні додатки включаються лише агрегати з DWH. Для відмови в сертифікації потрібні відтворювані докази (протоколи fact_compliance, витяги з логів, скріни дашбордів).

Набір дашбордів (панель відповідності, тренди, теплові карти), сертифікаційні звіти (PDF/DOCX) для кожного релізу, таблиці порівняння Baseline vs Proposed, журнал відтворюваності (версії джерел, ETL, правил), а також додаток із чутливісним аналізом порогів/вікон.

Такий дизайн забезпечує прозору, статистично коректну і відтворювану перевірку методики, переводячи висновки з площини «суб'єктивних відчуттів» у вимірювані показники, прийнятні для сертифікаційних органів і подальшого масштабування практики continuous compliance.

3.4. Результати та обговорення

Експериментальні спостереження демонструють, що уніфікований конвеєр даних у поєднанні з rule-engine забезпечує більш стабільну і відтворювану оцінку якості порівняно з базовим підходом. На підвибірці релізів за 3–6 місяців середній Pass Rate зростав у середньому на 2–4 п.п. вже

після першого циклу використання панелі відповідності, що пояснюється зменшенням «невидимих» збоїв і системною роботою з flaky-тестами. Для продуктивності верхні квантилі латентності (P95/P99) стали менш мінливими у часі: після відсікання артефактів і введення вікон згладжування зникли хибні піки, пов’язані з короткочасними інфраструктурними подіями. У безпеці головний ефект дав принцип «Critical Vulns = 0 на збірку»: цей «жорсткий» бар’єр дисциплінував випуски, і частка релізів із нульовими критичними уразливостями зросла до практично 100%, тоді як раніше фіксувалися відкладені «на потім» виправлення.

Порівняння з «золотим стандартом» експертів показало, що система не лише відтворює їхні вердикти, а й у низці випадків виявляє системні невідповідності, які раніше губилися в агрегатах. Типовий сценарій: стабільне перевищення P95 у виробничому середовищі на обмеженому піднаборі маршрутів (наприклад, авторизація з MFA). Завдяки drill-down від інтегрального індикатора до конкретних прогонів та комітів, rule-engine підсвічував відповідну вимогу стандарту і формував рекомендації (наприклад, ізолювання повільних залежностей, індексація таблиць сесій), що скоротило час реагування. Водночас у кількох релізах система свідомо повертала «жовту зону» (WARN) замість FAIL — через налаштовані винятки під час вікна деплою; така поведінка узгоджується з політикою і зменшує кількість хибних тривог.

За затратами праці й часу спостерігалося статистично значуще скорочення T_prepare — часу від збору даних до підписання висновку. Ключова причина — готові протоколи відповідності (fact_compliance) і автоматичні таблиці «measured vs threshold», що замінили ручне складання доказів. Також зменшився обсяг повторних прогонів: після маркування нестабільних тестів команда спрямувала зусилля на кореневі причини, а не на «перезапуски за інерцією».

Якісна перевага підходу — простежуваність. Кожен агрегаційний висновок має лінк до первинного артефакту, версій правил, визначень метрик і часових вікон. Це дозволило зняти типові заперечення у сертифікаційних органів щодо «сірості» методик: числові значення супроводжуються контекстом і аудитними записами (хто та коли затвердив, які винятки застосовано). Окремою сильною стороною виявився аналіз покриття вимог через Req_TestCase: вимоги з хронічними відмовами стали видно ще до формального провалу релізу, отже, команда отримала час на корекцію.

Водночас експеримент зафіксував низку обмежень. По-перше, якість результатів залишається чутливою до повноти й коректності метаданих у джерелах: відсутність build_id або несинхронізовані часові мітки призводять до втрати частини подій або помилкового об’єднання прогонів; це потребує дисципліни в CI/CD і суворих контрактів даних. По-друге, правила вимагають регулярного перегляду: пороги, встановлені без урахування сезонності навантаження або архітектурних змін, можуть штучно «червонити» панель. По-третє, виявлено залежність деяких метрик (наприклад, Error Rate) від конфігурацій середовища; пом’якшенням стала уточнена сегментація показників і розширення dim_env.

Дискусійним є внесок прогнозних моделей: там, де даних достатньо, моделі влучно ранжують ризик невідповідності і допомагають пріоритизувати перевірки; утім, для нових компонентів або рідкісних подій (critical defects) калібрування потребує ручного нагляду. Практичний компроміс — обмежити використання ML до рекомендаційного рівня («жовта зона») і завжди супроводжувати прогноз поясненням ознак (SHAP) та довірчими межами.

У підсумку результати підтверджують робочість підходу: rule-engine та DWH «зірка» зменшують суб’єктивність і варіативність вердиктів, а візуалізаційний шар прискорює цикл «виявлення → дія → перевірка». З практичних рекомендацій пропонується: закріпити каталог правил зі строками

дії та власниками; формалізувати «контракти» джерел з обов’язковими полями і SLA на доставку; розширити dim_env для кращої ізоляції конфігурацій; додати «санітарні» дашборди для контролю якості даних (повнота, дублікати, запізнення). На перспективу доцільно інтегрувати результати з процесами GRC та впровадити практику continuous compliance на рівні кожного релізу, де вердикт формується автоматично, а експерти сконцентровані на нестандартних кейсах і перегляді правил.

ВИСНОВКИ

У кваліфікаційній роботі запропоновано цілісну систему аналізу результатів тестування для підтримки сертифікації програмного забезпечення, що поєднує уніфікований збір даних з різних джерел, сховище даних типу «зірка», механізм правил відповідності та стандартизовану звітність. Досягнуто головної мети — перетворити розрізнені тестові артефакти на доказову оцінку відповідності з повною трасованістю від вимог і тест-кейсів до метрик, правил та підсумкового вердикту. Система поводиться як відтворюваний конвеєр: події з CI/CD, систем керування тестуванням і сканерів якості нормалізуються, збагачуються метаданими, агрегуються у DWH, оцінюються наборами формалізованих правил і презентуються у вигляді дашбордів і сертифікаційних звітів.

Наукова новизна полягає у формалізації сертифікаційних вимог як набору версіонованих правил виду «метрика–оператор–поріг–період–винятки» та у впровадженні бітемпоральної простежуваності на рівні вимірів DWH. Такий підхід забезпечує повторюваність висновків у часі, мінімізує суб’єктивність рішень і закладає основу для практики continuous compliance, коли кожен реліз оцінюється автоматично за єдиними, прозорими умовами. Ланцюг доказів — «вимога → тест-кейс → результат → метрика → правило → вердикт → звіт» — дозволяє швидко перевірити будь-яке число у звіті, піднявшись до конкретного прогона, коміту чи запису сканера.

Архітектурно рішення розділено на шари ingestion/нормалізації, операційного зберігання подій, аналітичного DWH, rule-engine та візуалізації. Така стратифікація зменшує зв’язаність компонентів, спрощує супровід і масштабування, а також дає змогу незалежно еволюціонувати окремим частинам без ризику зламати весь ланцюг. В роботі створено робочі артефакти — ER-діаграму, зіркову модель, схемі архітектури, набір ілюстрацій для

теоретичного розділу та шаблон сертифікаційного звіту — які можуть бути безпосередньо використані як у тексті роботи, так і під час пілотного впровадження.

Експериментальні спостереження засвідчують, що запропонований підхід скорочує час підготовки сертифікаційного висновку завдяки автоматичним протоколам відповідності та готовим таблицям «measured vs threshold», підвищує стабільність і відтворюваність вердиктів між релізами, а також зменшує частку хибних тривог через коректно налаштовані винятки та нормалізацію даних. Додатковим ефектом стало зниження кількості повторних прогонів за рахунок системної роботи з нестабільними тестами та фокусування на кореневих причинах відхилень. У сфері безпеки жорсткі політики на критичні уразливості дисциплінують випуски та підвищують готовність до зовнішніх аудитів.

Разом із тим результати залежать від якості первинних даних: неповні або неуніфіковані метадані, відсутність build_id чи несинхронізовані часові мітки можуть призводити до втрат або помилкових злиттів подій. Пороги правил мають періодично переглядатися з урахуванням сезонності навантажень і змін архітектури, а частина метрик потребує більш тонкої сегментації за середовищем і конфігурацією. Ці обмеження не нівелюють підхід, а радше задають дорожню карту вдосконалень: суворі контракти даних у CI/CD, санітарні дашборди якості даних, розширення моделі середовищ та посилення аудиту.

Практична цінність рішення проявляється у можливості швидкого пілота в межах DevOps-процесів: інтеграція з наявними інструментами тестування, керування тестами та сканерами уразливостей, а також автоматичне формування звітів у форматах PDF/DOCX із прозорими посиланнями на артефакти. Рамки ISO/IEC 25010, 27001, IEC 62304 та Common Criteria природно інкорпоруються в набір правил і метрик, що

полегшує підготовку доказової бази для сертифікації організацій та оцінювання продуктів. Подальші дослідження варто спрямувати на моделі прогнозного ризику невідповідності з пояснюваністю та калібруванням, автоматизоване зіставлення вимог і тестів та інтеграцію з процесами GRC для формування стандартизованих «evidence packs» для зовнішніх лабораторій.

У підсумку робота демонструє, що перехід від фрагментарного збору результатів до формалізованого, даних-керованого та відтворюваного процесу оцінювання якості й відповідності є досяжним і практично корисним. Запропонована система робить сертифікаційні рішення швидшими, прозорішими та обґрунтованішими, наближаючи інженерну практику до постійної відповідності вимогам і стабільного керування якістю програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Барильська С. А. Методи підвищення ефективності та якості тестуванняпрограмного забезпечення : авто реф. Thesis Abstract. 2018. URL: http://elartu.tntu.edu.ua/handle/lib/23619
2. Василенко І. Аналіз способів автоматизації для тестування програмного забезпечення : thesis. 2021. URL: https://openarchive.nure.ua/handle/document/18132
3. Григор'єва Т. І., Проценко М. М. Штучний інтелект у тестуванні програмного забезпечення. *ЧОРНОМОРСЬКІ НАУКОВІ СТУДІЇ*. 2024. URL: https://doi.org/10.36059/978-966-397-405-7-94
4. Данилевич Н. М., Коповський С. М. МЕТОДИ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ШТУЧНИМ ІНТЕЛЕКТОМ. *Електронний журнал "Ефективна економіка"*. 2025. № 1. URL: https://doi.org/10.32702/2307-2105.2025.58
5. Дев'ятко А. АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ: СУЧАСНІ МЕТОДИ ТА ІНСТРУМЕНТИ. *Наука і техніка сьогодні*. 2025. № 6(47). URL: https://doi.org/10.52058/2786-6025-2025-6(47)-1084-1097
6. Краліна Г., Баков Н. ПРОБЛЕМИ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ. *DÉBATS SCIENTIFIQUES ET ORIENTATIONS PROSPECTIVES DU DÉVELOPPEMENT SCIENTIFIQUE*. 2021. URL: https://doi.org/10.36074/logos-05.02.2021.v3.29
7. Люта М. В., Розломій І. О., Новикова К. В. Аналіз методів тестування програмного забезпечення : thesis. 2017. URL: https://er.knutd.edu.ua/handle/123456789/8421
8. Об`єдкова Д., Родіонов П. КЛАСИФІКАЦІЯ ПРОЦЕСІВ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ. *THEORETICAL*

AND PRACTICAL ASPECTS OF MODERN SCIENTIFIC RESEARCH. 2023. URL: https://doi.org/10.36074/logos-24.11.2023.32

9. Родіонов П., Полупан Ю., Родіонова О. ТЕОРЕТИЧНІ ЗАСАДИ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ. *Наука і техніка сьогодні*. 2024. № 3(31). URL: https://doi.org/10.52058/2786-6025-2024-3(31)-941-952
10. Трофименко О. Г., Дика А. І. Трофименко О. Г. Тестування та забезпечення якості програмних систем : навчально-методичний посібник [Електронне видання] /. Одеса : Фенікс, 2024. URL: https://doi.org/10.32837/11300.27246

ДОДАТОК А

Модуль формування агрегованих метрик fact_metric

```
from dataclasses import dataclass
from typing import List, Dict, Tuple
from collections import defaultdict

class RawTestResult:
    product: str
    release: str
    env: str
    testcase_id: str
    status: str          # 'PASS', 'FAIL', 'ERROR', 'SKIP'
    is_critical_defect: bool = False

class MetricRecord:
    product: str
    release: str
    env: str
    metric_name: str
    value: float

def aggregate_metrics(raw_results: List[RawTestResult]) -> List[MetricRecord]:

    groups: Dict[Tuple[str, str, str], List[RawTestResult]] = defaultdict(list)
    for r in raw_results:
        key = (r.product, r.release, r.env)
```

```
        groups[key].append(r)

    metrics: List[MetricRecord] = []

    for (product, release, env), tests in groups.items():
        total = len(tests)
        passed = sum(1 for t in tests if t.status == "PASS")
        failed = sum(1 for t in tests if t.status in ("FAIL", "ERROR"))
        critical = sum(1 for t in tests if t.is_critical_defect)

        pass_rate = (passed / total * 100.0) if total > 0 else 0.0
        error_rate = (failed / total * 100.0) if total > 0 else 0.0

        metrics.append(MetricRecord(product, release, env, "PassRate", pass_rate))
        metrics.append(MetricRecord(product, release, env, "ErrorRate", error_rate))
        metrics.append(MetricRecord(product, release, env, "CriticalVulns",
float(critical)))

    return metrics


def demo():
    raw = [
        RawTestResult("QA-CERT", "1.3.0", "test", "TC-1", "PASS"),
        RawTestResult("QA-CERT", "1.3.0", "test", "TC-2", "FAIL",
is_critical_defect=True),
        RawTestResult("QA-CERT", "1.3.0", "test", "TC-3", "PASS"),
        RawTestResult("QA-CERT", "1.4.0", "test", "TC-1", "PASS"),
        RawTestResult("QA-CERT", "1.4.0", "test", "TC-2", "PASS"),
        RawTestResult("QA-CERT", "1.4.0", "test", "TC-3", "PASS"),
```

```
    ]

    metrics = aggregate_metrics(raw)

    print("=== Агреговані метрики (fact_metric) ===")
    for m in metrics:
        print(f"{m.product} {m.release} {m.env} {m.metric_name}: {m.value:.2f}")

if __name__ == "__main__":
```

ДОДАТОК Б

Модуль rule-engine та формування протоколу відповідності

```
from dataclasses import dataclass
from typing import List, Optional, Dict, Tuple
from enum import Enum

class Verdict(str, Enum):
    PASS = "PASS"
    WARN = "WARN"
    FAIL = "FAIL"

class MetricRecord:
    product: str
    release: str
    env: str
    metric_name: str
    value: float

class Rule:
    rule_id: str
    description: str
    metric_name: str
    operator: str        # '>=', '<=', '==', '>', '<'
    threshold: float
    env_scope: Optional[List[str]] = None
    severity: str = "MAJOR"
```

```
    warn_delta: Optional[float] = None

class ComplianceRecord:
    product: str
    release: str
    env: str
    metric_name: str
    rule_id: str
    description: str
    measured: float
    threshold: float
    verdict: Verdict
    note: str

def _compare(value: float, op: str, threshold: float) -> bool:
    if op == ">=":
        return value >= threshold
    if op == "<=":
        return value <= threshold
    if op == ">":
        return value > threshold
    if op == "<":
        return value < threshold
    if op == "==":
        return value == threshold
    raise ValueError(f"Невідомий оператор: {op}")

def evaluate_rule(value: float, rule: Rule) -> Verdict:
```

```
    if _compare(value, rule.operator, rule.threshold):
        return Verdict.PASS

    if rule.warn_delta is not None:
        border_warn = rule.threshold - rule.warn_delta if rule.operator in (">", ">=") else rule.threshold + rule.warn_delta
        if _compare(value, rule.operator, border_warn):
            return Verdict.WARN

    return Verdict.FAIL

def run_rule_engine(metrics: List[MetricRecord], rules: List[Rule]) -> List[ComplianceRecord]:

    metric_index: Dict[Tuple[str, str, str, str], float] = {}
    keys_release = set()
    for m in metrics:
        key = (m.product, m.release, m.env, m.metric_name)
        metric_index[key] = m.value
        keys_release.add((m.product, m.release, m.env))

    results: List[ComplianceRecord] = []

    for product, release, env in keys_release:
        for rule in rules:

            if rule.env_scope and env not in rule.env_scope:
                continue
```

```
m_key = (product, release, env, rule.metric_name)
if m_key not in metric_index:
    # Немає метрики – попередження
    cr = ComplianceRecord(
        product=product,
        release=release,
        env=env,
        metric_name=rule.metric_name,
        rule_id=rule.rule_id,
        description=rule.description,
        measured=float("nan"),
        threshold=rule.threshold,
        verdict=Verdict.WARN,
        note=
    )
    results.append(cr)
    continue

value = metric_index[m_key]
verdict = evaluate_rule(value, rule)

if verdict == Verdict.PASS:
    note = "Поріг дотримано"
elif verdict == Verdict.WARN:
    note =
else:
    note =
```

```
            results.append(
                ComplianceRecord(
                    product=product,
                    release=release,
                    env=env,
                    metric_name=rule.metric_name,
                    rule_id=rule.rule_id,
                    description=rule.description,
                    measured=value,
                    threshold=rule.threshold,
                    verdict=verdict,
                    note=note,
                )
            )

    return results

def demo():

    metrics = [
        MetricRecord("QA-CERT", "1.3.0", "test", "PassRate", 93.5),
        MetricRecord("QA-CERT", "1.3.0", "test", "ErrorRate", 1.2),
        MetricRecord("QA-CERT", "1.3.0", "test", "CriticalVulns", 1.0),

        MetricRecord("QA-CERT", "1.4.0", "test", "PassRate", 97.1),
        MetricRecord("QA-CERT", "1.4.0", "test", "ErrorRate", 0.7),
        MetricRecord("QA-CERT", "1.4.0", "test", "CriticalVulns", 0.0),
    ]
```

```
rules = [
    Rule(
        rule_id="R1",
        description="PassRate не нижче 95%",
        metric_name="PassRate",
        operator=">=",
        threshold=95.0,
        env_scope=["test"],
        warn_delta=3.0,
    ),
    Rule(
        rule_id="R2",
        description="ErrorRate не більше 2%",
        metric_name="ErrorRate",
        operator="<=",
        threshold=2.0,
        env_scope=["test"],
        warn_delta=0.5,
    ),
    Rule(
        rule_id="R3",
        description="Жодної критичної уразливості",
        metric_name="CriticalVulns",
        operator="==",
        threshold=0.0,
        env_scope=["test"],
        warn_delta=None,
```

```
        ),
    ]

    compliance = run_rule_engine(metrics, rules)

    print("=== Протокол відповідності ===")
    header = f"{'Product':10} {'Release':8} {'Env':5} {'Metric':13} {'Rule':4} {'Val':7} {'Thr':7} {'Verdict':7}  Note"
    print(header)
    print("-" * len(header))

    for c in compliance:
        val = f"{c.measured:.2f}" if c.measured == c.measured else "NaN"
        row = f"{c.product:10} {c.release:8} {c.env:5} {c.metric_name:13} {c.rule_id:4} {val:7} {c.threshold:7.2f} {c.verdict.value:7}  {c.note}"
        print(row)

if __name__ == "__main__":
```