https://www.sworldjournal.com/index.php/swj/article/view/swj35-01-093
DOI: 10.30888/2663-5712.2026-35-01-093

УДК 004.05

RESEARCH OF THE VALIDITY BOUNDARIES OF INTEGRATION TESTS IN A LOCALLY REPRODUCED CLOUD EXECUTION ENVIRONMENT

ДОСЛІДЖЕННЯ МЕЖ ВАЛІДНОСТІ ІНТЕГРАЦІЙНИХ ТЕСТІВ У ЛОКАЛЬНО ВІДТВОРЕНОМУ ХМАРНОМУ СЕРЕДОВИЩІ ВИКОНАННЯ

Supryhan V.A. / Суприган В.А.
c.t.s. / к.т.н.
ORCID: 0000-0003-2539-8003
Lebid O.V. / Лебідь О.В.
PhD in economics / доктор філософії з економіки
ORCID: 0000-0003-2285-0543
Vinnytsia National Agrarian University, Vinnytsia, 3 Soniachna St., 21008
Вінницький національний аграрний університет, Вінниця, вул. Сонячна, 3, 21008

Анотація. *У сучасних cloud-native системах тестування програмного забезпечення дедалі більше залежить від характеристик середовища виконання, у якому запускаються інтеграційні перевірки. На відміну від класичних підходів до контролю якості, де середовище тестування вважається відносно стабільним, у хмарних архітектурах конфігурації, залежності, мережеві маршрути та механізми безпеки можуть істотно впливати на результати тестів, змінюючи їхню інтерпретацію та доказову силу. Загальні принципи якості програмного забезпечення і тестування, викладені у базових працях з QA, не завжди враховують специфіку багаторівневих хмарних середовищ виконання.*

Особливої уваги потребує практика локального відтворення хмарного середовища виконання, яка використовується для прискорення циклу розроблення та ізоляції інтеграційних перевірок. Такий підхід дозволяє зменшити залежність від спільних хмарних стендів, однак водночас створює ризик отримання результатів тестування, що лише частково відображають реальну поведінку системи у цільовому середовищі. У класичних моделях тестування, зокрема у контексті інтеграційних і системних перевірок, питання валідності результатів за різних умов виконання розглядається обмежено або опосередковано.

Метою статті є дослідження меж валідності інтеграційних тестів, виконаних у локально відтвореному хмарному середовищі, та формалізація підходів до оцінювання доказовості таких результатів у процесі контролю якості програмного забезпечення. У роботі обґрунтовано, що різні класи тестів мають різну чутливість до відмінностей середовища виконання, що узгоджується з сучасними уявленнями про стратегії тестування у складних програмних системах.

Основним результатом дослідження є запропонована модель оцінювання валідності тестових результатів на основі аналізу відповідності ключових характеристик середовища виконання, зокрема функціональної поведінки, конфігурації, залежностей, мережевих умов, безпеки та даних. Показано, що локально відтворене середовище може забезпечувати доказові результати для окремих класів інтеграційних перевірок, водночас залишаючись індикативним або недоказовим для інших. Отримані висновки створюють підґрунтя для більш усвідомленого використання локального відтворення хмарних середовищ у практиці тестування та зниження ризику хибних рішень у процесах забезпечення якості програмного забезпечення.

Ключові слова: *інтеграційне тестування, хмарне середовище виконання, локальне відтворення середовища, валідність результатів тестування, відповідність середовища (parity), контроль якості програмного забезпечення.*

Вступ.

У сучасних cloud-native системах тестування програмного забезпечення виконується в умовах багаторівневої інфраструктури, де середовище виконання стає невід'ємним чинником формування результатів перевірок [1]. На відміну від традиційних монолітних застосунків, для яких середовище тестування часто є відносно стабільним і легко контрольованим, хмарні сервіси залежать від складної сукупності конфігураційних параметрів, зовнішніх залежностей, мережевих маршрутів і механізмів безпеки, що є характерною ознакою сучасних хмарних технологій [2]. У таких умовах коректність і доказовість результатів тестування визначається не лише якістю самих тестових сценаріїв, а й ступенем відповідності середовища, у якому вони виконуються, цільовій моделі роботи системи [3].

Практика розроблення хмарних сервісів свідчить, що один і той самий набір інтеграційних або системних тестів може демонструвати різні результати залежно від того, де саме він запускається: у локальному середовищі розробника, у спільному cloud-dev або в стабілізованому інтеграційному стенді. Такі відмінності зумовлені розбіжностями у конфігураціях, складі та поведінці залежних сервісів, мережевих обмеженнях, моделях доступу до ресурсів і роботі механізмів автентифікації та авторизації, що є типовим викликом для повностекового тестування розподілених систем [4]. Внаслідок цього результати тестування можуть бути спотворені впливом факторів, не пов'язаних безпосередньо з перевірюваними змінами в коді, що ускладнює їх інтерпретацію та знижує довіру до тестових висновків [5].

Особливої актуальності проблема доказовості тестування набуває в контексті локального відтворення хмарного середовища виконання, яке широко використовується для прискорення циклу розроблення та ізоляції інтеграційних перевірок. Локально відтворені середовища дозволяють зменшити залежність від спільних хмарних стендів, уникнути взаємних блокувань між розробниками та знизити рівень шуму у вигляді нестабільних або флуктуючих результатів тестів, що відповідає сучасним практикам ефективного тестування в інженерних

командах [6]. Водночас таке відтворення, як правило, є частковим і передбачає спрощення або модифікацію окремих аспектів інфраструктури, зокрема мережевої моделі, механізмів безпеки та роботи з секретами.

За відсутності чітких критеріїв оцінювання адекватності локально відтвореного середовища виникає ризик формування хибних висновків щодо якості програмного забезпечення. У таких випадках тести можуть успішно проходити в локальному середовищі, створюючи ілюзію готовності змін до інтеграції або релізу, тоді як у стабілізованих хмарних середовищах аналогічні сценарії завершуються помилками або відмовами. Це свідчить про необхідність розмежування понять корисності тестового сигналу та його доказовості, а також про потребу формалізованого підходу до оцінювання валідності результатів тестування залежно від умов їх виконання, що особливо важливо для інтеграційного та компонентного тестування [7].

У зв’язку з цим актуальним є дослідження меж валідності інтеграційних тестів, виконаних у локально відтвореному хмарному середовищі. Таке дослідження має враховувати не лише технічні аспекти відтворення інфраструктури, а й специфіку різних класів тестів, ступінь їхньої залежності від окремих характеристик середовища та можливість коректної інтерпретації отриманих результатів у контексті загального процесу контролю якості програмного забезпечення, що узгоджується з сучасними підходами до підготовки фахівців і практиків тестування [8]. Водночас у практиці розроблення та перевірки хмарних сервісів важливу роль відіграють також дисципліни, орієнтовані на інженерний цикл створення змін і їх контроль, зокрема підходи, що виникають у контексті тест-орієнтованої розробки [9], а також інструментальні засоби автоматизації перевірок, які дозволяють формувати швидкий та відтворюваний зворотний зв’язок під час виконання інтеграційних тестів у локальних середовищах [10].

Постановка задачі.

У cloud-native системах середовище виконання не є нейтральним фоном для запуску тестів, а виступає активним фактором, що визначає поведінку сервісів і

коректність результатів перевірок. Практична потреба у швидкому інтеграційному зворотному зв'язку для розробника часто реалізується через локальне відтворення хмарного середовища виконання: запуск модифікованого сервісу локально разом із релевантною підмножиною залежностей, конфігурацій і маршрутів доступу, а також з'єднання з частиною хмарної інфраструктури через захищені канали або port-forward. Така організація дозволяє зменшити конкуренцію за спільні ресурси dev/sit, уникнути взаємного впливу паралельних деплоїв і підвищити керованість локального циклу розроблення.

Водночас локально відтворене середовище майже завжди є наближенням до хмарного еталона, а не його повною копією. Зміни або спрощення у конфігураціях, складі залежностей, мережевій моделі, правилах доступу та управлінні секретами можуть спотворювати тестові сигнали, породжуючи хибнопозитивні або хибнонегативні результати. За таких умов виникає методологічна проблема: навіть якщо тестові сценарії реалізовані коректно, залишається відкритим питання, чи мають отримані результати доказову силу для прийняття інженерних рішень (наприклад, щодо готовності змін до інтеграції або переходу на наступну стадію життєвого циклу).

Отже, задача даного дослідження полягає у формалізації підходу до оцінювання валідності результатів інтеграційних тестів, виконаних у локально відтвореному хмарному середовищі виконання, з урахуванням того, що різні класи тестів мають різну чутливість до відмінностей середовища. Для розв'язання цієї задачі необхідно:

- визначити поняття зворотного (reverse) підходу як способу формування локального тестового середовища на основі хмарної конфігурації;

- ввести та уточнити поняття parity як міри відповідності ключових характеристик середовища виконання;

- визначити валідність результатів тестування як ступінь доказовості висновків, які можуть бути зроблені на основі результатів запуску тестів у конкретному середовищі.

Основні терміни та визначення.

Зворотний підхід (reverse) до формування середовища виконання. Під reverse-підходом у межах даної роботи розуміється спосіб організації тестового середовища, за якого вихідною точкою є вже розгорнуте хмарне середовище (наприклад, dev або sit), а його конфігурація та поведінкові властивості використовуються як еталон для локального відтворення умов виконання сервісу або функціонально релевантної підмножини цих умов. Ключовою ознакою reverse-підходу є спрямованість процесу «від хмари до локального середовища»: локальна конфігурація, набір залежностей і маршрути доступу формуються так, щоб наблизити локальне виконання до хмарного еталона, зберігаючи при цьому ізоляцію від спільних середовищ та можливість швидкого тестування змін.

Локальне відтворення хмарного середовища виконання. Локальне відтворення у даному контексті означає запуск одного або кількох сервісів локально (наприклад, у контейнерах або локальному Kubernetes) разом із пов’язаними конфігураціями та залежностями, необхідними для виконання інтеграційних перевірок. Відтворення є частковим: частина інфраструктурних компонентів може залишатися у хмарі (керовані бази даних, брокери повідомлень, централізовані механізми безпеки), а доступ до них забезпечується через port-forward, VPN або інші тунельні механізми. Таким чином, reverse-підхід реалізує не повне дублювання хмарного середовища, а практично достатню проєкцію, орієнтовану на конкретний сценарій тестування.

Parity (відповідність середовища) у межах дослідження трактується як ступінь відповідності локально відтвореного середовища виконання хмарному еталону за визначеними вимірами, що впливають на коректність результатів тестування. На відміну від загального поняття «ідентичності середовищ», parity фокусується на тих характеристиках, які є значущими для конкретних тестових сценаріїв та їх інтерпретації. Для формалізації parity у роботі використовується набір вимірів відповідності:

Functional parity — відповідність функціональної поведінки сервісу та його інтеграційних контрактів очікуваній поведінці у хмарному еталоні.

Configuration parity — відповідність конфігураційних параметрів, що впливають на виконання коду (змінні середовища, конфігураційні файли, feature flags, параметри клієнтів залежностей).

Dependency parity — відповідність складу та поведінки зовнішніх залежностей (БД, брокери, кеші, пошукові сервіси, сторонні API), включно з версіями, режимами роботи та обмеженнями.

Network parity — відповідність мережевих умов: маршрутизації, рівнів проксі, затримок, DNS/Service Discovery, політик мережевої ізоляції та доступності сервісів.

Security parity — відповідність моделі безпеки: автентифікації, авторизації, ролей, політик доступу, сертифікатів, секретів і механізмів їх ротації.

Data parity — відповідність структури та характеристик даних: схеми, обмежень цілісності, репрезентативності тестових наборів, наявності необхідних довідників і станів.

Важливо, що parity не обов'язково передбачає максимальне відтворення за всіма вимірами. У практиці reverse-підходу допустимі цілеспрямовані спрощення, однак вони повинні бути зафіксовані та враховані при оцінюванні валідності результатів тестування.

Валідність результатів тестування (validity) у даній роботі визначається як ступінь доказовості результатів тестування, отриманих у конкретному середовищі виконання, для формування коректних висновків щодо якості програмного забезпечення у цільовому хмарному середовищі (SIT/UAT/Prod). Іншими словами, валідність відповідає на питання: чи можна на підставі отриманого результату робити обґрунтований висновок про поведінку системи в еталонному середовищі, і з яким рівнем довіри?

У межах статті валідність розглядається як категоріальна оцінка, що може набувати таких рівнів:

Valid (V) — результат є доказовим: ключові виміри parity для даного класу тестів збережені, і висновок може використовуватися для прийняття рішень.

Partially valid (PV) — результат є умовно доказовим: тестовий сигнал

корисний для раннього зворотного зв'язку, однак потребує підтвердження у стабілізованому середовищі через наявні відмінності parity.

Not valid (NV) — результат недоказовий: деградація критичних вимірів parity робить висновок некоректним або ризикованим, а використання такого результату може призвести до хибних рішень.

Таким чином, валідність виступає функцією двох факторів: (1) класу тестів і його чутливості до характеристик середовища; (2) реального рівня parity локально відтвореного середовища відносно хмарного еталона. Це створює основу для подальшої побудови матриці валідності, яка дозволяє формалізувати межі застосовності інтеграційних тестів у reverse-середовищах та підвищити керованість процесів контролю якості в cloud-native розробленні.

Таксономія тестів.

У межах даного дослідження інтеграційне тестування розглядається як багаторівневий набір перевірок, які відрізняються не лише предметом контролю (логіка сервісу, взаємодія компонентів, дані чи безпека), але й чутливістю до характеристик середовища виконання. У cloud-native системах, де архітектура будується на мережевій взаємодії сервісів, інфраструктурних залежностях та правилах доступу, доцільно застосовувати практичну таксономію тестів, що дає можливість розмежувати класи перевірок за типом їхньої доказовості у локально відтвореному середовищі. Такий підхід узгоджується з сучасними уявленнями про стратегії тестування, які підкреслюють необхідність співвідносити тип тесту з його цілями та контекстом виконання [3].

1.API Contract tests (контрактні тести API). Контрактні тести призначені для перевірки узгодженості інтерфейсів взаємодії між сервісами. У практиці cloud-native систем такими контрактами виступають OpenAPI-специфікації, схеми запитів/відповідей або consumer-driven contracts, що фіксують очікування клієнтів від поведінки сервісу. Основною метою є підтвердження сумісності контракту при зміні логіки сервісу, версій API або формату даних, що знижує ризик прихованих інтеграційних регресій. Контрактні тести часто можуть бути валідними у локально відтвореному середовищі, оскільки критично залежать від

функціональної поведінки та конфігураційних параметрів сервісу, а не від повної імітації мережевої чи безпекової моделі хмари [2].

2.Service Integration tests (інтеграційні тести сервісу з реальними залежностями). Цей клас тестів спрямований на перевірку коректності взаємодії сервісу з його залежностями: базами даних, брокерами повідомлень, кешами, файловими сховищами, пошуковими сервісами або зовнішніми API. На відміну від контрактних тестів, тут фокус зміщується на практичні аспекти інтеграції: формат даних, транзакційність, обробку помилок, повторні спроби, часові обмеження та коректність конфігурацій клієнтів. Такі тести є ключовими для підтвердження коректної роботи сервісу у складі реальної інфраструктури, що відповідає принципам повностекового тестування, де значення надається реальним інтеграційним ланцюжкам, а не лише ізольованим перевіркам [5].

3.Workflow / E2E tests (наскрізні сценарні тести). Наскрізні (end-to-end) тести перевіряють завершені сценарії взаємодії через кілька сервісів, включно з оркестрацією бізнес-процесів, обробкою подій, побудовою ланцюгів запитів і реакцією системи на зміни стану. Їхня цінність полягає у підтвердженні того, що система працює як цілісна одиниця з позиції кінцевого користувача або споживача API. Однак саме цей клас тестів зазвичай є найбільш чутливим до невідповідностей середовища виконання, оскільки потребує високого рівня відповідності залежностей, даних, мережевої поведінки та безпекових механізмів. Практичні стратегії тестування підкреслюють, що E2E-перевірки мають найвищу доказову силу лише за умов близької відповідності цільовому середовищу та стабільності інтеграційного стенда [6].

4.Security / Authorization tests (перевірки безпеки та авторизації). Даний клас тестів зосереджений на перевірці правил доступу, ролей, політик безпеки, scope-моделей, обмежень на рівні сервісів, а також коректності реакцій системи на несанкціоновані дії. У cloud-native системах ці перевірки тісно пов’язані з механізмами автентифікації та авторизації, обробкою токенів, сертифікатів і ролей, а також із зовнішніми системами управління ідентичністю. Важливою особливістю цього класу тестів є критична залежність від security parity: якщо

локальне середовище спрощує або “моккає” безпеку, результати можуть втрачати доказовість і формувати хибну впевненість щодо коректності доступів у хмарі. У практичних принципах побудови надійної автоматизації тестування акцентується, що перевірки безпеки мають виконуватися у максимально наближених до реальних умовах [7].

5.Resilience tests (перевірки стійкості та деградацій). Тести стійкості спрямовані на оцінювання поведінки системи у випадках часткових відмов, затримок, нестабільності залежностей або тимчасової деградації сервісів. До них належать перевірки таймаутів, повторних спроб (retries), circuit breaker-механізмів, fallback-логіки, а також контроль коректності обробки помилок на межі між сервісами. У cloud-native системах, де відмови є очікуваним класом подій, resilience-перевірки мають критичну залежність від мережевих характеристик (network parity) та поведінки залежностей у реальних умовах. Разом з тим, у локально відтвореному середовищі можливе виконання “chaos-lite” перевірок як швидкого індикатора коректності захисної логіки, за умови коректної інтерпретації результатів та усвідомлення обмежень такого тестового стенда [8].

6.Data/Schema migration tests (перевірки міграцій та сумісності схем). Цей клас тестів забезпечує контроль змін структури даних: міграцій баз даних, еволюції схем, сумісності форматів повідомлень, обмежень цілісності та коректності перетворень даних. Для cloud-native сервісів, які активно розвиваються та масштабуються, перевірка міграцій є критичною, оскільки помилки у схемах можуть проявлятися на інтеграційних стадіях або вже в продукції. Значущість цього класу перевірок посилюється тим, що він часто може бути достатньо валідним у локально відтвореному середовищі за умови контрольованої data parity (наявності репрезентативної схеми, тестових наборів та обмежень), що дозволяє виявляти помилки ще до інтеграції змін у спільні стенди. Ідея раннього підтвердження коректності змін через контрольовані цикли розробки та тестування також узгоджується з підходами тест-орієнтованої розробки як засобу зниження ризику дефектів [9].

Модель валідності (O / UO / OP + V / PV / NV).

Для формалізації цієї залежності пропонується використовувати модель валідності, яка базується на пріоритизації вимірів відповідності середовища виконання та на категоріальній шкалі оцінювання доказовості отриманого результату. Модель дозволяє перейти від інтуїтивного оцінювання (“тут ніби схоже на хмару”) до керованого підходу, де кожен результат тестування може бути інтерпретований з урахуванням того, які умови виконання були забезпечені та які обмеження існували під час запуску перевірок.

Задамося такими пріоритети вимірів відповідності: O / UO / OP. Оскільки різні класи тестів по-різному залежать від характеристик середовища, для кожного класу доцільно визначати пріоритети вимірів parity у вигляді трьох груп:

- обов’язкові (O) — такі характеристики середовища виконання, без яких тестовий результат втрачає доказову силу або стає систематично спотвореним. Дефіцит відповідності за O-вимірами означає, що тест не може виступати надійним доказом коректності системи в цільовому хмарному середовищі;

- умовно обов’язкові (UO) — характеристики, які істотно підвищують доказовість результатів і знижують ризик хибних сигналів, однак у деяких сценаріях допускають обмежене спрощення за умови коректної інтерпретації отриманих результатів. Відсутність або деградація UO-вимірів зазвичай переводить тестовий сигнал із доказового рівня у частково валідний;

- опційні (OP) — характеристики, що покращують точність та наближеність середовища до еталона, але не є критичними для формування базового висновку в межах конкретного класу тестів. OP-виміри можуть впливати на зручність, стабільність або деталізацію перевірок, однак їх відсутність рідко робить результат недоказовим.

Таким чином, для кожного класу тестів формується “профіль залежності від середовища”, який описує, які саме виміри parity є критичними для коректної інтерпретації результатів. Практична користь такого підходу полягає у можливості заздалегідь оцінити очікуваний рівень доказовості результатів

тестування ще до виконання тестів, виходячи з параметрів локально відтвореного середовища.

Пропонуємо модель валідності, яка може бути застосована у вигляді простого правила інтерпретації:

- якщо порушено хоча б один O-вимір, результат слід маркувати як NV;
- якщо всі O-виміри збережено, але є суттєві відхилення за UO-вимірами — результат слід маркувати як PV;
- якщо всі O-виміри збережено, а UO-виміри або збережені, або мають неістотні відмінності — результат може вважатися V.

Відмінності за OP-вимірами не змінюють клас валідності напряму, але можуть бути зафіксовані як фактори, що впливають на точність, стабільність і повторюваність тестових результатів.

У такий спосіб модель валідності дозволяє формалізувати межі доказовості інтеграційних тестів у локально відтвореному хмарному середовищі та забезпечити керований перехід від “інтуїтивного” тестового фідбеку до підходу, де результати перевірок мають прозору інтерпретацію та чітко визначений рівень довіри. Це створює підґрунтя для побудови матриці валідності, у якій класи тестів зіставляються з пріоритетами O/UO/OP за вимірами parity та відповідними рівнями V/PV/NV.

Матриця валідності.

Після введення моделі валідності (V / PV / NV) та визначення пріоритетів вимірів відповідності середовища виконання (Обов’язкові — O, Умовно обов’язкові — UO, Опційні — OP) доцільним є побудова узагальнювального артефакту, який дозволяє швидко та формалізовано оцінювати доказовість результатів інтеграційних тестів у локально відтвореному хмарному середовищі. Таким артефакт виступає матриця валідності, що поєднує класи тестів із їх “профілями залежності” від parity-вимірів та типовим рівнем валідності у reverse-підході.

Матриця виконує дві функції. По-перше, вона виступає як інструмент узгодження термінів та очікувань між розробниками й QA: для кожного класу

тестів фіксується, які характеристики середовища є критично необхідними, а які можуть бути частково спрощені. По-друге, матриця є практичним механізмом маркування довіри до результатів тестування, коли прийняття рішення (інтегрувати зміни чи ні) залежить не тільки від факту “pass/fail”, але й від того, наскільки отриманий результат може вважатися доказом якості в умовах цільового хмарного стенду (SIT/UAT).

Таблиця 1 - Матриця валідності тестів у локально відтвореному середовищі

Клас тестів	Functional	Configuration	Dependency	Network	Security	Data	Типова валідність у reverse
API Contract tests	O	O	UO	OP	UO	OP	V / PV
Service Integration tests	O	O	O	UO	UO	UO	PV
Workflow / E2E tests	O	O	O	O	O	O	PV / NV
Security / Authorization tests	UO	UO	OP	UO	O	OP	NV (за mocked security)
Resilience tests	O	O	UO	O	UO	OP	PV
Data/Schema migration tests	UO	O	UO	OP	OP	O	V / PV

Авторська розробка

1.API Contract tests (таблиця 1). Контрактні тести мають високу залежність від Functional та Configuration parity, оскільки їх ціль — перевірка узгодженості API-поведінки та форматів взаємодії Водночас повна відповідність мережевої моделі або security-механізмів не завжди є критичною для валідації контракту, що дозволяє отримувати доказові або частково валідні результати у reverse-

середовищі. Ризик зниження валідності виникає тоді, коли авторизаційні механізми впливають на сам контракт (наприклад, різні відповіді залежно від ролі або scope), у такому випадку Security переходить із UO у O.

2.Service Integration tests (таблиця 1). Інтеграційні тести сервісу з реальними залежностями найчастіше потребують високої відповідності Dependency parity, а також стабільної конфігурації та коректної функціональної поведінки. Однак у reverse-підході частина інфраструктури зазвичай доступна через port-forward/VPN, що може спотворювати мережеві властивості, а також призводити до часткових спрощень security. Тому типовий рівень валідності для цього класу оцінюється як PV, оскільки результат є корисним для раннього фідбеку, але потребує підтвердження у стабілізованому QA-середовищі.

3.Workflow / E2E tests (таблиця 1). Наскрізні перевірки залежать від повного ланцюга компонентів і їх взаємодії, тому для них критичними є всі виміри parity: функціональність, конфігурація, залежності, мережа, безпека та дані. У локально відтвореному середовищі практично складно забезпечити повну відповідність усім цим вимірам без суттєвої інфраструктурної вартості. Тому E2E-тести у reverse-середовищі часто мають лише часткову валідність, а в багатьох випадках — недоказові результати, якщо середовище істотно відрізняється за network/security/data характеристиками.

4.Security / Authorization tests (таблиця 1). Перевірки безпеки та авторизації є найбільш критичними до Security parity, оскільки саме модель доступу, ролі, політики та робота секретів формують предмет тестування. Якщо локальне середовище застосовує mocked security або тестові спрощені токени, результат тестів, навіть якщо він “позитивний”, не може бути доказом коректності у хмарі. У reverse-сценаріях ці тести можуть зберігати індикативну цінність лише за умови cloud-backed моделі безпеки (коли авторизація фактично залишається хмарною), однак у типовому випадку вони оцінюються як NV.

5.Resilience tests (таблиця 1). Тести стійкості потребують високої відповідності Network parity, оскільки саме мережеві властивості (затримки, часткові відмови, нестабільність каналів) і поведінка залежностей визначають

реакції системи. У reverse-середовищі можна виконувати “chaos-lite” перевірки — наприклад, штучно вводити таймаути або переривати з’єднання на рівні клієнта, однак це не гарантує повної відповідності поведінки в Kubernetes-мережі чи service mesh. Тому валідність зазвичай PV: корисний сигнал про наявність захисної логіки, але не фінальний доказ стійкості.

6.Data/Schema migration tests (таблиця 1). Тести міграцій і сумісності схем є чутливими до Data parity та Configuration parity. Якщо локально забезпечено репрезентативну схему БД, типові обмеження цілісності та коректні міграції, цей клас тестів може демонструвати високу валідність навіть у reverse-середовищі. Зниження валідності виникає переважно тоді, коли тестові дані спрощені настільки, що не відображають ключових структур або умов продукційного середовища, або коли реальні залежності замінені на несумісні аналоги.

Матриця валідності може використовуватися як операційний інструмент QA у вигляді простого правила: перед запуском тестів команда визначає, який клас перевірок виконується, після чого зіставляє реальні властивості локального середовища з O/UO/OP-вимірами. На цій основі тестові результати маркуються рівнем V/PV/NV, що дозволяє:

- ✓ зменшити ризик хибних рішень через некоректні QA-сигнали;
- ✓ фіксувати очікуваний рівень довіри до тестового прогону ще до запуску;
- ✓ формувати прозорі правила переходу від локального фідбеку до обов’язкової верифікації у SIT/UAT;
- ✓ стандартизувати інтерпретацію результатів тестування у командах із високою паралельністю розроблення.

Таким чином, матриця валідності забезпечує системну основу для оцінювання доказовості інтеграційних тестів у локально відтворених середовищах і створює передумови для подальшого аналізу кейсів, у яких результати тестування є лише частково валідними або недоказовими через відмінності у ключових характеристиках середовища виконання.

Кейси.

Для практичного обґрунтування меж валідності інтеграційних тестів у

локально відтвореному хмарному середовищі доцільно розглянути типові ситуації, у яких результати тестування набувають статусу PV (частково валідні) або NV (недоказові). Наведені нижче приклади не претендують на універсальність, однак вони відображають поширені патерни розбіжностей між локальним reverse-середовищем і контрольними хмарними стендами, зокрема SIT/UAT. У кожному кейсі показано, які саме відмінності parity-вимірів спричиняють спотворення тестового сигналу та чому коректна інтерпретація результату потребує маркування рівнем довіри.

Кейс 1. Service Integration: порт-форвардинг до керованої бази даних → PV через Network parity.

Ситуація. Розробник запускає сервіс локально (reverse) та підключає його до хмарної керованої бази даних через kubectl port-forward або VPN-тунель. Інтеграційні тести перевіряють типові CRUD-операції, транзакції та бізнес-інваріанти на рівні сервісу.

Спостереження. Частина тестів може демонструвати нестабільність: періодичні таймаути, випадкові збої з'єднання, повільні відповіді або помилки повторного підключення. При цьому в SIT аналогічний набір тестів проходить стабільно.

Пояснення причини. У reverse-середовищі функціональна логіка сервісу та конфігурація (Functional/Configuration parity) можуть бути збережені на достатньому рівні, а залежність (Dependency parity) навіть може бути "реальною" (керована БД у хмарі). Однак шлях доступу до залежності відрізняється від Kubernetes-моделі: порт-форвардинг вводить додатковий проксі-рівень, змінює мережеві характеристики та збільшує мінливість затримок. Це означає деградацію Network parity, яка стає ключовою причиною нестабільності результатів.

Висновок щодо валідності. Результати таких інтеграційних тестів варто трактувати як PV: вони корисні для раннього виявлення регресій у логіці чи схемах даних, але не можуть бути єдиним доказом "готовності" змін без підтвердження у стабілізованому середовищі.

Кейс 2. Security/Authorization: локально підмінені секрети та ролі у NV через Security parity.

Ситуація. У reverse-середовищі сервіс запускається локально з підставними токенами або спрощеною авторизаційною моделлю (наприклад, вимкнено перевірку підпису JWT, використано статичний “admin” токен або замінено інтеграцію з Identity Provider). Тести перевіряють доступ до endpoint’ів залежно від ролей і дозволів (RBAC, scopes, policies).

Спостереження. Тести проходять успішно локально, але у SIT/UAT ті самі сценарії завершуються помилками “403 Forbidden”, “insufficient permissions”, відмовами доступу до ресурсів або некоректною маршрутизацією запитів через політики безпеки.

Пояснення причини. У цьому випадку локальне середовище може підтримувати правильну бізнес-логіку endpoint’ів (Functional parity) і навіть зберігати більшість конфігураційних параметрів (Configuration parity). Проте предмет тесту — модель безпеки, і саме вона є обов’язковою умовою доказовості результату. Якщо security-механізми спрощені, підмінені або “замокані”, то тест фактично перевіряє інший об’єкт, ніж той, що існує у хмарі. Це спричиняє false confidence: ілюзію коректності доступів при фактичній невідповідності продукційним правилам.

Висновок щодо валідності. Такі результати необхідно маркувати як NV, оскільки деградація Security parity стосується обов’язкового (O) виміру для даного класу тестів. Локальний прогін у такому режимі може використовуватися лише як допоміжна перевірка функціональності, але не як підтвердження коректності авторизації.

Кейс 3. Workflow/E2E: локально відтворена підмножина сервісів у PV або NV через Dependency/Data parity.

Ситуація. У reverse-середовищі розробник запускає локально лише змінюваний сервіс і частину найближчих залежностей, а інші компоненти або відсутні, або підмінені спрощеними заглушками. E2E-тести перевіряють сценарій “від запиту до завершеної бізнес-операції”, що в реальній системі

включає брокер повідомлень, асинхронну обробку та оновлення станів у кількох сервісах.

Спостереження. Локально E2E-сценарій може проходити швидко і стабільно, однак у SIT виникають відмінності: асинхронні події обробляються інакше, таймінги стають непередбачуваними, з'являються гонки станів, або система демонструє некоректну поведінку через відсутність певного сервісу/черги/топіка чи через іншу структуру даних.

Пояснення причини. У даному випадку ключовою проблемою є часткове відтворення системи. Навіть якщо функціональність окремого сервісу вірна, E2E-перевірка залежить від повної композиції компонентів, що включає Dependency parity (наявність реальних брокерів, черг, конфігурації ретраїв), а також Data parity (коректні стани, довідники, структури даних, які впливають на гілки бізнес-логіки). Якщо залежності не відтворені або дані "спрощені", тест або перевіряє лише частину процесу (що дає PV), або фактично не перевіряє потрібний сценарій взагалі (що дає NV).

Висновок щодо валідності. Якщо критичні сервіси та дані відсутні, або підмінені заглушками без еквівалентної поведінки — результат слід маркувати як NV.

Якщо відтворена більшість ключових залежностей, але відсутні окремі нефункціональні умови (реальні таймінги, повний набір даних) — результат може бути PV як корисний індикатор, що потребує підтвердження у SIT/UAT.

Узагальнення спостережень.

Наведені кейси демонструють, що PV/NV у reverse-середовищі виникає не через "погані тести", а через обмеження відповідності середовища виконання. Найпоширеніші причини спотворення тестових сигналів полягають у:

- деградації Network parity при тунелюванні до хмарних залежностей;
- спрощенні Security parity через підміну секретів або авторизаційної моделі;
- частковій реплікації системи, що впливає на Dependency та Data parity у сценарних перевірках.

Таким чином, кейси підтверджують доцільність маркування результатів

тестування рівнями V/PV/NV та використання матриці валідності як інструмента коректної інтерпретації QA-сигналів у процесі локального відтворення хмарного середовища виконання.

Рекомендації для процесу QA.

Запровадження локально відтвореного хмарного середовища виконання (reverse-підхід) доцільно розглядати як інструмент прискорення інтеграційного зворотного зв'язку, а не як повну заміну стабілізованих контрольних стендів SIT/UAT. Практична цінність reverse-середовища полягає у можливості швидко перевіряти зміни в ізольованому контексті без впливу на спільні ресурси команди, однак доказовість отриманих результатів має визначатися з урахуванням класу тестів і рівня відповідності середовища (parity). Тому першочерговою рекомендацією є розмежування тестових прогонів за їх призначенням: локальні перевірки мають забезпечувати раннє виявлення дефектів і регресій, тоді як фінальні рішення щодо готовності змін повинні спиратися на результати тестування у контрольному середовищі.

У reverse-середовищі доцільно запускати ті класи перевірок, які мають найвищу валідність за умов часткового відтворення хмарної інфраструктури. Насамперед це контрактні тести API та частина інтеграційних тестів сервісу із залежностями, оскільки вони переважно залежать від функціональної поведінки та коректної конфігурації, а не від повного відтворення мережевої моделі або політик безпеки. Також ефективними є тести міграцій та сумісності схем, за умови забезпечення релевантної структури даних, обмежень цілісності та контрольованого набору тестових даних. Водночас наскрізні E2E-сценарії та перевірки авторизації/безпеки мають запускатися в reverse-середовищі лише як індикативні, або ж переноситися на SIT/UAT як обов'язковий етап підтвердження, оскільки вони найбільш чутливі до network/security/data parity. Для перевірок стійкості reverse-середовище може бути корисним як "chaos-lite" індикатор коректності захисної логіки (timeouts, retries), однак не як остаточний доказ поведінки системи під реальними мережевими умовами.

Щоб уникнути некоректних висновків, результати тестування в reverse-

середовищі доцільно маркувати рівнем довіри відповідно до моделі V/PV/NV. Якщо для класу тестів забезпечені всі обов'язкові (O) parity-виміри, результат може вважатися V (доказовим). Якщо обов'язкові виміри збережені, але є деградація умовно обов'язкових (UO) — результат слід трактувати як PV (частково валідний) і підтверджувати у SIT/UAT. Якщо порушено хоча б один обов'язковий вимір (наприклад, security parity у тестах авторизації) — результат має вважатися NV (недоказовим) і не може бути підставою для інженерного рішення. Практично це може реалізовуватися через короткий "QA-артефакт відповідності", де фіксується: які залежності були локальними, які — хмарними; які секрети замінено; які мережеві компроміси застосовано (VPN/port-forward); та який рівень валідності очікується для основних груп тестів.

Висновки.

У роботі обґрунтовано, що локальне відтворення хмарного середовища виконання є ефективним механізмом зменшення взаємних блокувань у команді та прискорення циклу інтеграційного тестування, однак валідність отриманих результатів є нерівномірною та залежить від класу тестів і збережених parity-вимірів. Запропоновано підхід до оцінювання доказовості результатів на основі (1) таксономії тестів, (2) моделі пріоритизації характеристик середовища за групами O/UO/OP та (3) категоріальної шкали валідності V/PV/NV. Побудована матриця валідності дозволяє формалізувати межі застосовності reverse-середовища як джерела QA-сигналів та пояснити типові ситуації формування частково валідних або недоказових результатів.

Подальші дослідження доцільно спрямувати на експериментальну верифікацію запропонованої моделі шляхом порівняння результатів тестування в reverse-середовищі та у контрольному стенді SIT/UAT з використанням вимірюваних показників узгодженості результатів і частоти хибних сигналів. Перспективним є розроблення практичного механізму автоматизованого маркування тестових прогонів рівнем валідності на основі аналізу конфігурацій, складу залежностей і параметрів безпеки, а також інтеграція такого маркування в CI-процеси як додаткового критерію прийняття рішень щодо якості змін.

Література:

1.Якість програмного забезпечення та тестування: базовий курс. Навчальний посібник / За ред. Крепич С.Я., Співак І.Я. / для бакалаврів галузі знань 12 «Інформаційні технології» спеціальності 121 «Інженерія програмного забезпечення». – Тернопіль: ФОП Паляниця В.А., 2020. – 478 с.

2.Rex Black, Dorothy Graham, Erik van Veenendaal Foundations of Software Testing ISTQB Certification, 4th edition. Cengage Learning EMEA, 2020. – 288 p.

3.Matthew Heusser, Michael Larsen Software Testing Strategies: A testing guide for the 2020s. Packt Publishing, 2023. – 378 p.

4.Хмарні технології: навч. посіб. / [О.В. Зінченко, С.М. Іщеряков, С.В. Прокопов, С.О. Сєрих, В.В. Василенко]. – К: ФОП Гуляєва В. М., 2020. – 74 с.

5.Gayathri Mohan Full Stack Testing. A Practical Guide for Delivering High Quality Software. O'reilly Media, 2022. – 406 p.

6.Mauricio Aniche Effective Software Testing: A developer's guide. Manning, 2022. – 328 p.

7.Vladimir Khorikov Unit Testing Principles, Practices, and Patterns: Effective testing styles, patterns, and reliable automation for unit testing, mocking, and integration testing with examples in C#. Manning, 2020. – 304 p.

8.Kristin Jackvony The Complete Software Tester: Concepts, Skills, and Strategies for High-Quality Testing. Kindle Edition, 2021. – 514 p.

9.Saleem Siddiqui Learning Test-Driven Development: A Polyglot Guide to Writing Uncluttered Code 1st Edition. O'reilly Media, 2021. – 280 p.

10.Brian Okken Python Testing with pytest. Simple, Rapid, Effective, and Scalable. 2nd Edition. Pragmatic Bookshelf, 2022. – 274 p.

Abstract. *In modern cloud-native systems, software testing increasingly depends on the characteristics of the execution environment in which integration checks are performed. Unlike traditional quality assurance approaches, where the test environment is often considered relatively stable, cloud architectures involve complex configurations, external dependencies, network routing, and security mechanisms that can significantly influence test outcomes and their interpretation. General principles of software quality and testing, as presented in foundational QA literature, do not always fully address the impact of multi-layered cloud execution environments on the evidential value of test results.*

Particular attention is required for the practice of locally reproducing cloud execution environments, which is widely used to accelerate the development feedback loop and isolate integration testing activities. While this approach reduces reliance on shared cloud-based development or integration environments, it simultaneously introduces the risk that test results obtained locally may only partially reflect the actual behavior of the system in the target cloud environment. In classical testing models, especially in the context of integration and system testing, the issue of result validity across different execution environments is often treated implicitly or insufficiently formalized.

The purpose of this study is to investigate the boundaries of validity of integration tests executed in locally reproduced cloud execution environments and to formalize an approach for assessing the evidential strength of such results within software quality assurance processes. The paper argues that different classes of tests exhibit varying sensitivity to discrepancies in execution environments, which aligns with contemporary views on testing strategies for complex software systems.

The main outcome of the research is a proposed model for evaluating the validity of test results based on the analysis of environment parity across key dimensions, including functional behavior, configuration, dependencies, network conditions, security mechanisms, and data characteristics. It is demonstrated that locally reproduced environments can provide evidential test results for certain classes of integration checks, while remaining only indicative or non-evidential for others. The findings establish a foundation for more informed use of local cloud environment reproduction in testing practice and for reducing the risk of misleading quality signals in software engineering decision-making.

Key words: *integration testing, cloud execution environment, local environment reproduction, test result validity, environment parity, software quality assurance.*

Статтю надіслано: 26.01.2026 г.