https://doi.org/10.31891/2219-9365-2026-86-9
УДК 004.415.53

МЕЛЬНИК Андрій
Тернопільський національний технічний університет імені Івана Пулюя
https://orcid.org/0009-0003-6222-5598
e-mail: andrii.melnyk.it@gmail.com
ДМИТРОЦА Леся
Тернопільський національний технічний університет імені Івана Пулюя
https://orcid.org/0000-0003-2583-3271
e-mail: dmytrotsa.lesya@gmail.com

МЕТОДИ ТА АРХІТЕКТУРНІ ПІДХОДИ ДО АВТОМАТИЗАЦІЇ ТЕСТУВАННЯ МОБІЛЬНИХ І ВЕБЗАСТОСУНКІВ

У статті представлено аналіз методів автоматизації тестування програмного забезпечення з акцентом на мобільні та вебзастосунки. Досліджено сучасні методології забезпечення якості, проаналізовано архітектурні підходи фреймворків автоматизації та оцінено їхню ефективність відповідно до міжнародного стандарту ISO/IEC 25010:2023. Актуальність дослідження зумовлена необхідністю скорочення циклів розробки за умови збереження високої надійності та зручності використання цифрових продуктів. Здійснено порівняльний аналіз еволюції підходів до тестування та запропоновано формалізований метод вибору архітектури автоматизації. Запропонований підхід відрізняється тим, що враховує взаємозв'язок характеристик проєкту, доступних ресурсів і технічних обмежень при визначенні доцільної архітектури. Наукова новизна одержаних результатів полягає у розробці алгоритму прийняття рішень, що базується на кількісній оцінці метрик проєкту та забезпечує більш обґрунтований вибір рішень у процесі планування тестування. Отримані результати дозволяють зменшити технічні ризики та підвищити ефективність використання ресурсів при підтримці тестової інфраструктури.

Ключові слова: автоматизація тестування, архітектура тестового фреймворку, мобільні та вебзастосунки, забезпечення якості, метод вибору архітектури.

MELNYK Andrii, DMYTROTSA Lesia
Ternopil Ivan Pului National Technical University

METHODS AND ARCHITECTURAL APPROACHES TO AUTOMATING THE TESTING OF MOBILE AND WEB APPLICATIONS

The research presents a comprehensive analysis of software testing automation, specifically focusing on mobile and web applications. While manual testing remains common, its limitations – such as low repeatability and high labor intensity – often lead to errors and increased development costs. Thus, automated testing is essential for accelerating functional verification and optimizing expenses in highly competitive markets.

The primary aim of this study is to enhance quality assurance (QA) efficiency by developing a scientifically grounded method for the rational selection of an automation architecture. The methodology examines modern QA practices, architectural patterns, and the ISO/IEC 25010:2023 standard. This standard defines product quality through eight core characteristics, forming specific requirements for the test framework architecture to ensure reliable validation.

A comparative analysis of Linear, Modular, Data-Driven (DDF), Keyword-Driven (KDF), and Behavior-Driven (BDD) frameworks was conducted using expert evaluations across implementation speed, support cost, flexibility, skill requirements, and ROI. Acknowledging that no classic pattern is universally applicable, the study introduces an adaptive architecture selection method based on the mathematical formalization of input project parameters. The algorithm correlates framework selection with expected development duration, team expertise, business requirements, and data variability. This allows engineers to make deterministic decisions during planning, mitigating risks.

The developed method was validated by modeling architectural solutions for MVP/Prototypes, FinTech/Banking systems, and E-commerce CMS. Implementing this approach transitions QA from intuitive tool selection to engineering-based design. This prevents the accumulation of technical debt, effectively reducing the Total Cost of Ownership (TCO) of the automation system by 30-40% over the long term. Ultimately, test automation serves as a strategic risk management tool, resolving the trade-off between time-to-market and product quality.

Keywords: test automation, test framework architecture, mobile and web applications, quality assurance, architecture selection method.

Стаття надійшла до редакції / Received 27.03.2026
Прийнята до друку / Accepted 23.04.2026
Опубліковано / Published 31.05.2026

ПОСТАНОВКА ПРОБЛЕМИ У ЗАГАЛЬНОМУ ВИГЛЯДІ ТА ЇЇ ЗВ'ЯЗОК ІЗ ВАЖЛИВИМИ НАУКОВИМИ ЧИ ПРАКТИЧНИМИ ЗАВДАННЯМИ

Стрімке зростання вимог до продуктивності та надійності мобільних і вебзастосунків змушує IT-індустрію шукати нові шляхи оптимізації процесів забезпечення якості. Проте традиційні підходи до ручного контролю програмного забезпечення мають суттєві обмеження: вони характеризуються низькою повторюваністю, що робить їх економічно недоцільними в умовах безперервного розгортання [1]. Попри це,

великий обсяг перевірок досі виконується людьми. Це неминуче призводить до помилок через людський фактор, сповільнює випуск нових версій та збільшує загальні витрати проєкту. Відповідно, впровадження автоматизації стає критичною необхідністю для прискорення перевірки функціоналу та оптимізації бюджету.

Спеціалізовані програмні засоби здатні точно відтворювати дії користувача, виконуючи рутинні перевірки без участі тестувальника [2]. Такий підхід докорінно змінює процес забезпечення якості, спонукаючи компанії масово переходити на автоматизовані методи. Швидкість виконання, висока точність результатів та мінімізація помилок роблять автоматизацію стандартом у цій галузі [3, 4].

На практиці ці інструменти беруть на себе виконання заздалегідь визначених дій, автоматично порівнюють отримані результати з очікуваними та генерують деталізовані звіти [5]. Головна перевага цього підходу полягає у його економічній доцільності: розроблений одного разу тестовий сценарій можна легко повторювати та масштабувати без додаткових витрат часу. Відтворити подібне охоплення під час ручного тестування вкрай складно.

ФОРМУЛЮВАННЯ ЦІЛЕЙ СТАТТІ

Метою роботи є: підвищення ефективності процесів забезпечення якості мобільних та вебзастосунків шляхом розробки методу обґрунтованого вибору архітектури автоматизації. Досягнення цієї мети передбачає вирішення комплексу завдань, що включають аналіз існуючих підходів, виокремлення ключових метрик ефективності та побудову моделі прийняття рішень. У цьому контексті дослідження фокусується на процесах автоматизованого тестування, а безпосереднім предметом вивчення є методи та архітектурні моделі, що забезпечують оптимізацію витрат та надійності програмних продуктів.

Характеристики якості мобільних та вебзастосунків

Успіх програмного продукту залежить не лише від його базової функціональності, а й від відповідності суворим нефункціональним вимогам. Тому для контролю якості необхідний єдиний системний підхід, який спирається на стандартизовані показники ефективності та надійності.

Основою для побудови стратегії тестування виступає модель якості продукту, регламентована сучасним міжнародним стандартом ISO/IEC 25010:2023 (System and software Quality Requirements and Evaluation – SQuaRE), який замінив застарілу модель ISO/IEC 9126 [6]. Цей стандарт визначає якість продукту як ступінь задоволення заявлених та передбачуваних потреб користувачів, декомпозуючи її на вісім характеристик, кожна з яких формує специфічні вимоги до архітектури тестового фреймворку.

Ключовою характеристикою є функціональна придатність, що визначає ступінь відповідності функцій системи потребам користувачів. Автоматизовані функціональні тести, зокрема регресійні, виступають основним інструментом для перевірки бізнес-логіки. У тісному взаємозв'язку з нею перебуває продуктивність, яка характеризує поведінку системи у часі відносно використаних ресурсів. Для мобільних застосунків критичними показниками є час відгуку та енергоефективність. Об'єктивна оцінка цих параметрів під навантаженням вимагає застосування автоматизованих інструментів профілювання, здатних імітувати різні стани мережі та апаратних ресурсів.

Через велику різноманітність апаратного забезпечення на ринку, критичного значення набувають характеристики сумісності та здатності до перенесення. Вони визначають здатність продукту функціонувати в гетерогенних середовищах та обмінюватися даними з іншими системами. Автоматизація дозволяє нівелювати ризики несумісності шляхом використання ферм пристроїв для паралельного виконання тестів на різних конфігураціях.

Критичними для стабільності системи є надійність та захищеність. Надійність передбачає стійкість до збоїв та здатність до відновлення, що перевіряється шляхом імітації розривів з'єднання та помилок API. Захищеність же гарантує конфіденційність та цілісність даних, що вимагає інтеграції автоматизованих сканерів вразливостей у процеси безперервної інтеграції (CI/CD).

Завершують модель якості характеристики зручності використання та зручності супроводу. І якщо перша стосується ефективності взаємодії користувача з інтерфейсом, автоматизація якої включає перевірку навігації та доступності інтерфейсу, то зручність супроводу безпосередньо впливає на архітектуру самого тестового фреймворку. Низька придатність тестового коду до супроводу призводить до непропорційного зростання витрат на підтримку, що з часом може нівелювати економічний ефект від впровадження автоматизації. Таким чином, імплементація стандарту ISO/IEC 25010 дозволяє забезпечити комплексне тестове покриття, де автоматизація виступає не просто інструментом прискорення, а необхідною умовою перевірки багатовимірної якості програмного продукту.

Для формалізації процесу оцінювання якості кожній характеристиці стандарту ISO/IEC 25010 поставлено у відповідність набір кількісних метрик (KPI), які підлягають автоматизованому моніторингу (табл. 1).

Таблиця 1

Ключові метрики автоматизованого оцінювання якості

Характеристика якості	Метрика автоматизації	Одиниця виміру	Цільове значення
Функціональна придатність	Рівень успішності тестів	%	> 95%
Продуктивність	Час відгуку API	мс	< 200 мс
	Час запуску	с	< 2 с
Сумісність	Покриття пристроїв	%	Топ-80% ринку
Надійність	Частка сесій без збоїв	%	> 99.9%
Захищеність	Кількість критичних вразливостей (CVSS > 7)	шт.	0
	Час сканування	хв	< 15 хв (у конвеєрі)
Зручність використання	Accessibility Score (напр. Lighthouse)	бали	> 90/100
	Биті посилання	%	0%
Зручність супроводу	Покриття коду тестами	%	> 70-80%
	Цикломатична складність коду	од.	< 10-15
Портативність	Успішність встановлення/оновлення	%	100%

Архітектура та компоненти систем автоматизації

Сучасний фреймворк автоматизації тестування – це не монолітний інструмент, а комплексна система. Вона складається з окремих взаємопов'язаних компонентів, що дозволяє повторно використовувати код та значно спрощує технічний супровід. Архітектура такої системи повинна базуватися на принципах модульності та чіткого розподілу відповідальності між її складовими (рис. 1).

Рис. 1. Ключові компоненти фреймворку для автоматизації тестування

Щоб краще зрозуміти архітектуру, систему автоматизації (*SAS*) можна представити у вигляді моделі «чорної скриньки». Формально систему можна описати як відображення вхідних векторів у вихідні артефакти:

$$SAS: \{Sc, D, C\} \xrightarrow{Engine} \{R, L, A\}, \quad (1)$$

де *Sc* – код сценаріїв, *D* – тестові дані, *C* – конфігурація.

Процес виконання трансформує ці дані у вихідні результати: R (Звіти), L (Логи), A (Артефакти дефектів).

Основою будь-якого фреймворку є тестові сценарії. Вони містять команди, які імітують дії користувача та перевіряють реакцію програми. Щоб ці сценарії працювали ефективно, їх необхідно структурувати та об'єднати в логічні модулі. Це підвищує читабельність коду і допомагає інженерам швидко розуміти призначення кожної перевірки. Для уніфікації підходів до написання сценаріїв у межах команди необхідне впровадження стандартизованих рекомендацій, що регламентують правила найменування та використання архітектурних моделей, спрощуючи залучення нових фахівців до проєкту.

Ефективність виконання сценаріїв прямо корелює з якістю управління тестовими даними. Фреймворк повинен містити механізми централізованого зберігання та маніпулювання різними типами даних, включаючи граничні значення та некоректні вхідні параметри, що мінімізує ризик помилок. У випадках роботи з великими масивами інформації доцільним є використання підсистеми автоматичної генерації даних, що підвищує гнучкість тестування. Важливо також очищати дані після тестів, щоб система повернулася до початкового стану. При цьому для корпоративних систем обов'язковою є реалізація механізмів безпеки, що забезпечують захист конфіденційних даних та управління доступом під час тестування.

Безпосередня реалізація перевірок покладається на механізм виконання тестів та конфігурацію середовища. Модуль виконання координує всі процеси: планує запуски та розподіляє ресурси для паралельного виконання тестів. При цьому тестове середовище (сервери, бази даних, мережа) має максимально точно відтворювати реальні робочі умови застосунку. Для гарантування сумісності застосунку фреймворк повинен підтримувати крос-платформне та крос-браузерне тестування, виявляючи потенційні невідповідності роботи на різних пристроях. Стабільність цього процесу забезпечується механізмами обробки помилок та автоматичного відновлення, які дозволяють продовжувати виконання набору тестів навіть у разі локальних збоїв або мережевих проблем.

Завершальними компонентами архітектури є підсистеми звітності та глибока інтеграція із системами безперервного розгортання (CI/CD). Це забезпечує автоматичний запуск перевірок після кожної зміни коду, миттєвий зворотний зв'язок та швидку діагностику дефектів. Такий комплексний підхід, що охоплює стратегії підтримки та оновлення скриптів, дозволяє адаптувати процес тестування до еволюції програмного продукту та зменшити кількість збоїв у довгостроковій перспективі.

Таким чином, системна інтеграція окреслених компонентів формує архітектурний базис ефективного фреймворку автоматизації тестування мобільних та вебзастосунків. Їх узгоджена взаємодія забезпечує комплексний підхід до верифікації якості, дозволяючи стандартизувати процеси розробки сценаріїв, оптимізувати час виконання перевірок та гарантувати високу достовірність отриманих результатів, що є критичним фактором для прийняття обґрунтованих управлінських рішень щодо готовності програмного продукту до випуску.

Ефективність автоматизації значною мірою залежить від обраної стратегії тестування, яка визначає глибину доступу до внутрішньої структури системи та цілі верифікації. У науковій літературі домінує класифікація за рівнем доступу до коду (моделі «скриньок») та етапами життєвого циклу розробки.

Найбільш поширеним підходом в автоматизації мобільних та вебзастосунків є тестування «чорної скриньки» (англ. Black-box). На практиці такий підхід означає роботу безпосередньо з графічним інтерфейсом або відкритими API. Сценарії просто імітують дії людини і перевіряють, чи збігаються отримані результати з очікуваними, не заглиблюючись у внутрішню архітектуру коду [7]. Статистичний аналіз показує, що значна частина зусиль в автоматизації спрямована саме на цей тип тестування, оскільки він є потужним інструментом для оцінки реагування системи з точки зору кінцевого користувача [8].

На противагу цьому, тестування «білої скриньки» (англ. White-box) вимагає повного доступу до архітектури застосунку та базується на глибокому знанні внутрішньої структури коду [9]. Цей метод, який часто виконується безпосередньо розробниками, є вичерпним, але трудомістким, що обмежує його застосування в автоматизації системного рівня [10]. Стратегія «сірої скриньки» (англ. Grey-box) набуває особливого значення у мобільній автоматизації. Нативні інструментальні фреймворки (зокрема, XCUITest для екосистеми Apple та Espresso для Android) забезпечують тісну інтеграцію безпосередньо в середовище розробки (наприклад, Xcode). Це дозволяє тестам напряму взаємодіяти з ієрархією відображення, мінімізуючи проблему нестабільних тестів, притаманну класичним підходам «чорної скриньки» [11].

З точки зору цільового призначення, критично важливу роль в автоматизації відіграє регресійне тестування. Воно спрямоване на виявлення негативних побічних ефектів, що виникають внаслідок змін у коді. З економічної точки зору, автоматизація регресійного тестування вирішує проблему лінійного зростання витрат, притаманну ручному підходу. Оскільки обсяг регресійного набору збільшується з кожною ітерацією розробки, автоматизація дозволяє трансформувати змінні витрати у фіксовані інвестиції в інфраструктуру. Більше того, інтеграція сучасних механізмів ШІ-асистування (зокрема, самовідновлюваних тестових скриптів) дозволяє суттєво знизити вартість супроводу регресійних наборів, максимізуючи загальну економічну рентабельність проєкту [12].

Типологія та порівняльний аналіз фреймворків

Автоматизація тестування мобільних та вебзастосунків реалізується за допомогою спеціалізованого фреймворку, який забезпечує структуроване середовище для створення та виконання автоматизованих тестів. Фреймворк визначає чіткі критерії оцінки результатів тестування, знижує витрати на підтримку тестів і спрощує процес визначення очікуваної поведінки застосунків. Вибір відповідного фреймворку є вирішальним фактором для ефективного й успішного автоматизованого тестування.

Лінійний фреймворк – це базовий архітектурний підхід, який часто є першим кроком в автоматизації. Він ґрунтується на послідовному записі та детермінованому відтворенні дій користувача. На практиці цей метод часто реалізується через інструменти запису макросів, що дозволяє швидко генерувати тестові скрипти без необхідності глибокого занурення в програмування. Однак згенерований таким чином код є процедурним і містить жорсткі прив'язки до конкретних тестових даних та координат елементів інтерфейсу. Головним недоліком такої архітектури є надзвичайна чутливість до змін. Це унеможливлює повторне використання компонентів коду та робить підтримку тестів економічно невигідною на великих, довготривалих проєктах.

Архітектура модульного підходу безпосередньо базується на логічній декомпозиції системи, що тестується. Замість написання суцільних монолітних скриптів, застосунок, що тестується, логічно розділяють на окремі компоненти, модулі або функції, для яких створюються ізольовані скрипти перевірки. Такий підхід дозволяє сформувати додатковий шар абстракції, який приховує пряму технічну взаємодію з елементами інтерфейсу всередині окремих модулів. Це забезпечує значну стійкість тестів до змін: у разі оновлення ідентифікатора елемента на сторінці необхідно внести правки лише в один відповідний модуль, тоді як основна логіка тестових сценаріїв залишається незмінною. Хоча цей метод вимагає вищої кваліфікації інженерів на етапі проєктування архітектури, він суттєво спрощує подальше масштабування та технічний супровід тестового набору.

Фреймворк автоматизації тестування на основі даних передбачає відокремлення логіки тестових сценаріїв від самих тестових даних мобільних та вебзастосунків. Ця архітектура вирішує проблему дублювання логіки шляхом повної сепарації алгоритмів тестування від вхідних значень. Особливістю цього методу є параметризація: єдиний тестовий алгоритм ітеративно обробляє різні масиви даних, які система динамічно зчитує із зовнішніх джерел (баз даних чи конфігураційних файлів). Таке рішення забезпечує глибоке покриття граничних значень і класів еквівалентності, уникаючи при цьому надмірного розростання кодової бази. Використання цього підходу критично підвищує ефективність регресійного тестування, дозволяючи перевіряти сотні варіацій вхідних параметрів за допомогою єдиного універсального скрипта [13].

Натомість підхід до тестування за ключовими словами є системно-незалежним підходом, який базується на концепції тестування за допомогою зовнішніх наборів даних. В рамках такого підходу тестові сценарії створюються через спеціальні блоки коду, відомі як «ключові слова». Ключові слова, які визначають дії, необхідні для взаємодії з мобільними та вебзастосунками, зберігаються у зовнішніх табличних структурах. Тестові дані та ключові слова відокремлені від логіки виконання тестів і не залежать від конкретного інструмента автоматизації. Це дає можливість повторно використовувати код і суттєво зменшує вимоги до навичок програмування для тестувальників. Однак основною проблемою такого підходу є управління життєвим циклом ключових слів. Відповідно до стандарту ISO/IEC/IEEE 29119-5:2024, неконтрольоване розростання бібліотеки ключових слів призводить до надмірних витрат на рефакторинг та вимагає жорсткої регламентації з боку експертів з автоматизації [14].

Фреймворк розробки на основі поведінки наголошує на співпраці між технічними та нетехнічними членами команди, що працюють над застосунком. Цей підхід фокусується на перевірці бізнес-вимог продукту шляхом створення тестів, зрозумілих для всіх учасників процесу розробки, включаючи нетехнічних фахівців. Технічно це реалізується через створення проміжного шару трансляції між специфікаціями, описаними природною мовою, та виконуваним програмним кодом. Сценарії формулюються у форматі історій користувача, які автоматично перетворюються інструментами автоматизації на виклики відповідних функцій та методів драйвера. Така структура не лише покращує комунікацію в команді, але й дозволяє використовувати тестову документацію як живі специфікації системи, що завжди відповідають її реальному функціональному стану.

Щоб вибрати найкращу архітектуру для автоматизації тестування, потрібно врахувати кілька критеріїв одночасно. Для об'єктивного вибору досліджувані підходи зіставлено за п'ятьма ключовими параметрами: швидкістю розгортання, вартістю технічного супроводу, гнучкістю, вимогами до кваліфікації персоналу та довгостроковою економічною ефективністю.

Кількісний вимір цих показників ґрунтується на методі експертних оцінок. Основою для нього став синтез емпіричних даних із наукових публікацій та детальний аудит технічних обмежень інструментарію. Для формалізації результатів застосовано п'ятибальну дискретну шкалу. Найвищу оцінку отримують рішення з очевидними технічними перевагами, як-от найвища швидкодія чи гранична економічність супроводу. Мінімальне значення, навпаки, вказує на наявність критичних бар'єрів: неможливості розширення системи або невиправдано високої вартості її утримання. Отримані результати, які відображають кореляцію характеристик кожної архітектурної моделі з метриками якості, систематизовано у відповідній матриці (табл. 2).

Таблиця 2

Кількісна оцінка ефективності архітектурних підходів

Архітектурний підхід	Швидкість впровадження	Вартість підтримки	Гнучкість та масштабування	Вимоги до кваліфікації	Економічна ефективність
Лінійний	5	1	1	5	1
Модульний	3	4	4	3	4
На основі даних (DDF)	3	4	5	3	5
На основі ключових слів (KDF)	2	3	3	4	3
Керований поведінкою (BDD)	2	3	4	2	4

Науково-методичний підхід до вибору архітектури автоматизації

Проблема вибору архітектури автоматизації зводиться до задачі багатокритеріальної оптимізації. Її цільовою функцією визначено максимізацію довгострокової рентабельності за наявних ресурсних обмежень. Для формалізації цього процесу розроблено модель M, яка визначає відображення множини вхідних параметрів проєкту P на множину доступних архітектурних рішень A.

Функціональна залежність має вигляд:

$$M: P \rightarrow A, \tag{2}$$

Вектор вхідних параметрів *P* визначається кортежем:

$$P = \langle T_{dur}, S_{team}, R_{biz}, D_{var} \rangle, \tag{3}$$

де T_{dur}(Duration) – прогнозована тривалість проєкту (у місяцях), $T_{dur} \in (0, +\infty)$, S_{team}(Team Skills) – рівень компетенції команди, $S_{team} \in \{Junior, Middle, Senior/SDET\}$, R_{biz}(Business Requirement) – логічна змінна, що вказує на необхідність узгодження сценаріїв бізнес-аналітиками, $R_{biz} \in \{0,1\}$, D_{var} (Data Variability) – ступінь варіативності тестових даних, $D_{var} \in \{Low, High\}$.

Множина вихідних рішень A включає:

$$A = \{Linear, Modular, DDF, KDF, BDD, Hybrid\}, \tag{4}$$

Розроблений метод базується на стратегії послідовного виключення, що дозволяє усунути технічно неефективні варіанти ще на початкових стадіях проєктування. Процедуру алгоритмізовано у вигляді чотирьох послідовних умов.

На першому етапі аналізується часовий фактор (T_{dur}). Встановлено емпіричний поріг окупності автоматизації, що становить 2 місяці. Якщо $T_{dur} < 2$, інвестиції в розробку складної архітектури перевищують потенційну економію ресурсів ($ROI < 1$). За таких умов єдиним рентабельним варіантом залишається лінійний підхід, який гарантує максимально швидкий випуск продукту, попри ускладнений подальший технічний супровід.

Наступним кроком аналізуються комунікаційні вимоги (R_{biz}). Якщо $R_{biz} = 1$, простір пошуку автоматично звужується до методології керованої поведінкою (BDD). Цей підхід забезпечує семантичну прозорість тестів, виступаючи мостом між бізнес-вимогами та технічною реалізацією, хоча й збільшує накладні витрати на підтримку шару визначення кроків.

Далі оцінюється структура інформаційних потоків (D_{var}). Для систем, де бізнес-логіка залишається стабільною, але вимагає перевірки на широкому спектрі вхідних значень (комбінаторика даних), застосування імперативних скриптів призводить до дублювання коду. Оптимальним рішенням у такому контексті є фреймворк на основі даних (англ. Data-Driven Framework, DDF), що дозволяє відокремити тестові дані від логіки виконання, забезпечуючи покриття тисяч тестових випадків єдиним параметризованим сценарієм.

Четвертий етап враховує інженерну зрілість команди (S_{team}) для довгострокових проєктів. Якщо кваліфікація персоналу обмежена($S_{team} = Junior$), рекомендовано фреймворк на основі ключових слів (англ. Keyword-Driven Framework, KDF), який абстрагує складність коду за ключовими словами. Для зрілих команд ($S_{team} \geq Middle$) найбільш ефективними є модульна або гібридна архітектури, що забезпечують найвищий рівень інкапсуляції, повторного використання коду та інтеграції в системи безперервного розгортання (CI/CD).

Для наочного представлення запропонованого науково-методичного підходу розроблено UML-діаграму діяльності (рис. 2). Вона комплексно відображає процеси попереднього оцінювання характеристик фреймворків та покрокової алгоритмічної фільтрації простору рішень A на основі вхідного вектора параметрів проєкту P.

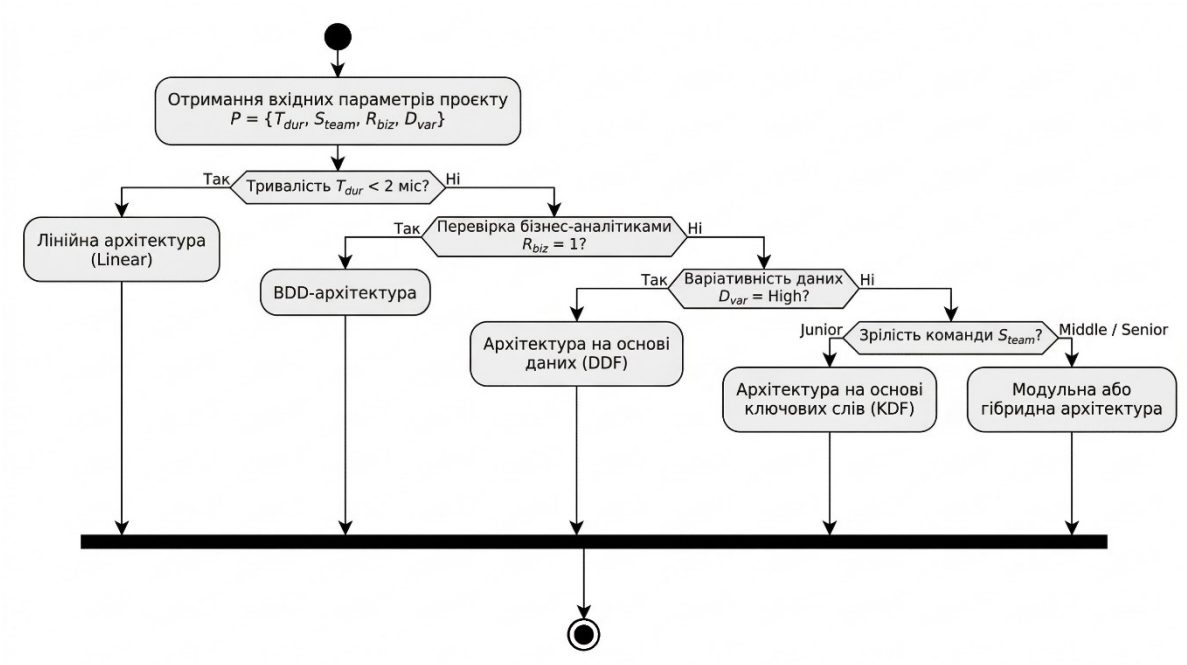

Рис. 2. UML-діаграма діяльності процесу оцінювання та відбору архітектури автоматизації тестування

Апробацію розробленого методу здійснено шляхом моделювання архітектурних рішень для трьох типових класів програмних продуктів. Зокрема, обрані для моделювання проєкти репрезентують різні масштаби розробки та специфічні галузеві вимоги: від швидких прототипів до складних фінансових систем. Такий підхід дозволив на практиці перевірити коректність роботи кожної гілки запропонованого алгоритму та переконатися у відсутності логічних суперечностей при комбінації різних значень вхідних параметрів. Результати аналізу, наведені в таблиці 3, підтверджують здатність методу адаптуватися до різних вхідних умов та доводять його придатність для використання в реальних умовах розробки.

Таблиця 3

Матриця вибору архітектури на основі розробленого алгоритму

Клас системи	Вектор параметрів P	Рекомендоване рішення A	Обґрунтування ефективності
Прототип	$T_{dur} = 1.5$; $S_{team} = Junior$; $D_{var} = Low$;	Лінійна	Мінімізація початкових інвестицій. Швидкий зворотний зв'язок на етапі запуску.
Банківські системи	$T_{dur} > 36$; $D_{var} = High$; $S_{team} = Senior$;	Гібридна (DDF + Модульна)	Забезпечення надійності при обробці транзакцій та масштабованості кодової бази на роки.
Платформи електронної комерції	$R_{biz} = 1$; $T_{dur} = 12$; $S_{team} = Middle$;	Керована поведінкою(BDD)	Тести виконують роль "живої документації", знижуючи ризики неправильної інтерпретації вимог замовника.

Цей метод допомагає замінити інтуїтивний вибір інструментів на науково обґрунтований підхід. Це об'єктивно мінімізує сукупну вартість володіння тестовою інфраструктурою, зменшуючи ризик накопичення технічного боргу під час довгострокового супроводу продукту.

ВИСНОВКИ З ДАНОГО ДОСЛІДЖЕННЯ І ПЕРСПЕКТИВИ ПОДАЛЬШИХ РОЗВІДОК У ДАНОМУ НАПРЯМІ

Аналіз засвідчив тенденцію до переходу від імперативних лінійних сценаріїв до декларативних моделей. Це гарантує абстрагування тестової логіки від технічної реалізації та дає змогу залучати до контролю якості нетехнічних учасників проєкту. Встановлено, що жоден із класичних підходів в ізольованому вигляді не є універсальним: прості рішення не масштабуються, а складні – можуть бути економічно невиправданими для малих проєктів.

З метою подолання невизначеності під час проєктування систем автоматизації, запропоновано метод адаптивного вибору архітектури, що спирається на математичну формалізацію вхідних параметрів.

Запропонована модель дозволяє обґрунтовано корелювати вибір фреймворку з очікуваною тривалістю розробки, рівнем кваліфікації команди, вимогами бізнесу та варіативністю даних. На основі цієї моделі сформульовано алгоритм послідовної фільтрації, що дозволяє інженерам приймати детерміновані рішення на етапі планування та уникати «парадоксу автоматизації».

Доведено, що ефективність автоматизації прямо залежить від стратегії управління тестовими даними та об'єктивного прогнозування економічної рентабельності. Апробація розробленого методу показала, що для короткострокових проєктів оптимальним залишається лінійний підхід, тоді як для довгострокових корпоративних систем найбільш доцільним є використання гібридних архітектур із зовнішнім управлінням даними. На практиці використання такого підходу дозволяє зменшити загальні витрати на підтримку системи.

У роботі доведено, що обґрунтований вибір архітектури автоматизації є ключовим фактором мінімізації технічних та фінансових ризиків проєкту. Використання метрик стандарту ISO/IEC 25010 у поєднанні із запропонованим архітектурним підходом дозволяє структурувати процес верифікації, охоплюючи не лише функціональність, а й надійність, безпеку та зручність використання.

References

1. Mridha, N. F., Keya, A. N. Automated web testing over the last decade: A systematic literature review. Int. J. Softw. Eng. Knowl. Eng. 2023, 33(05), 701–735.

2. Aebersold, K. Test Automation Framework.

3. Jamil, M.A.; Arif, M.; Abubakar, N.S.A.; Ahmad, A. Software testing techniques: A literature review. In Proceedings of the Proceedings – 6th International Conference on Information and Communication Technology for the Muslim World, ICT4M, Jakarta, Indonesia, 22–24 November 2016.

4. Muccini, H.; Informatica, D.; Di Francesco, A.; Informatica, D.; Esposito, P.; Informatica, D. Software Testing of Mobile Applica- tions: Challenges and Future Research Directions. In Proceedings of the 7th International Workshop on Automation of Software Test (AST), Zurich, Switzerland, 2–3 June 2012; pp. 29–35.

5. Tramontana, P.; Amalfitano, D.; Amatucci, N. Automated functional testing of mobile applications: A systematic mapping study. Softw. Qual. J. 2019, 149–201.

6. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) – Product quality model : ISO/IEC 25010:2023. – [Ed. 2.0]. – Geneva : International Organization for Standardization, 2023. – 22 p.

7. Tirodkar, A.A.; Khandpur, S.S. EarlGrey: iOS UI Automation Testing Framework. In Proceedings of the 2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft), Montreal, QC, Canada, 25–25 May 2019; pp. 12–15.

8. Vajak, D.; Grbic, R.; Vranjes, M.; Stefanovic, D. Environment for Automated Functional Testing of Mobile Applications. In Proceedings of the 2018 International Conference on Smart Systems and Technologies (SST), Osijek, Croatia, 10–12 October 2018; pp. 125–130.

9. Bansal, A. A Comparative Study of Software Testing Techniques. Int. J. Comput. Sci. Mob. Comput. 2014, 3, 579–584.

10. Kaur Chauhan, A.R.; Singh, B.A.I. Latest Research and Development on Software Testing Techniques and Tools. Int. J. Curr. Eng. Technol. 2014, 4, 2368–2372.

11. Yadav, N., Sharma, P. Top Mobile Testing Frameworks Compared: Appium, Espresso, XCUITest. In Proceedings of the 2025 IEEE International Conference on Software Testing, Verification and Validation (ICST), 2025; pp. 112–119.

12. Goyal, R., Kumar, A. A Multi-Year Grey Literature Review on AI-assisted Test Automation. IEEE Trans. Softw. Eng. 2024, 50(2), 214–230.

13. Alotaibi, S. The Applicability of Automated Testing Frameworks for Mobile Application Testing: A Systematic Literature Review. IEEE Access, 2023, 11, 45123-45135.

14. Software and systems engineering – Software testing – Part 5: Keyword-Driven Testing : ISO/IEC/IEEE 29119-5:2024. – Geneva : International Organization for Standardization, 2024. 48 p.