УДК 004.415.2
DOI https://doi.org/10.37406/2706-9052-2026-1-35

Лебідь О. В.
доктор філософії з економіки,
старший викладач кафедри комп'ютерних наук та цифрової економіки,
Вінницький національний аграрний університет
Вінниця, Україна
E-mail: *sshlebid@gmail.com*
ORCID: *0000-0003-4253-8696*

АВТОМАТИЗАЦІЯ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ: КЛЮЧОВІ МЕТОДОЛОГІЇ ТА ПІДХОДИ

Анотація

Розвиток напрямку тестування програмного забезпечення становить важливу складову сучасної інженерії програмних систем, яка має понад сімдесят років історії становлення та вдосконалення. Автоматизоване тестування як науково-прикладна дисципліна сьогодні відіграє ключову роль у підвищенні якості, надійності та стійкості програмних продуктів у різних сферах застосування. Це особливо актуально для національних інформаційних систем України, що функціонують в умовах цифрової трансформації, інтеграції з європейськими стандартами та посиленої потреби в захищених та безвідмовних рішеннях для державних, економічних і соціальних процесів.

У статті проведено системний аналіз і класифікацію основних видів тестування програмного забезпечення, включаючи функціональне, нефункціональне, модульне, інтеграційне, системне й регресійне тестування. Здійснено огляд сучасних підходів до автоматизації тестових процесів, що ґрунтуються на актуальних методологіях – зокрема, практиках Agile та DevOps, а також концепціях shift-left і shift-right, які забезпечують раннє виявлення дефектів і безперервну оцінку якості програмних модулів.

Особлива увага приділена опису рівнів автоматизованого тестування та технологічних засобів, що застосовуються у дослідженні: інструментальних фреймворків для автоматичного генерування тестів, середовищ безперервної інтеграції і доставки (CI/CD), а також практичній інтеграції з системами контролю версій. Проаналізовано переваги й обмеження автоматизованого тестування у контексті українських проєктів розробки програмного забезпечення, зокрема підкреслено важливість адаптації тестових стратегій до реалій обмежених ресурсів, динамічних змін вимог та необхідності оперативного впровадження змін.

Результати дослідження обґрунтовують, що впровадження автоматизованого тестування сприяє підвищенню ефективності інженерних процесів, мінімізації кількості помилок у продуктивних середовищах та зміцненню довіри до програмних продуктів як для внутрішніх користувачів, так і для міжнародних партнерів.

Ключові слова: *автоматизоване тестування, програмне забезпечення, якість програмного забезпечення, інформаційні системи, модульне тестування, методології тестування, автоматизація тестових процесів.*

Вступ. У сучасних умовах стрімкого розвитку цифрових технологій та зростання ролі інформаційних систем у функціонуванні економіки, державного управління й соціальної сфери питання забезпечення якості програмного забезпечення набуває особливої актуальності. Надійність, стабільність і безпека програмних продуктів є критичними чинниками ефективного функціонування як комерційних, так і державних інформаційних систем, особливо в умовах цифрової трансформації та інтеграції України до європейського інформаційного простору.

Однією з ключових складових процесу забезпечення якості програмного забезпечення є тестування, яке дозволяє виявляти дефекти, оцінювати відповідність програмних модулів функціональним і нефункціональним вимогам, а також мінімізувати ризики експлуатаційних збоїв. Із зростанням складності програмних систем та динамічності вимог традиційні підходи до ручного тестування втрачають ефективність, що зумовлює активний розвиток і впровадження автоматизованого тестування.

В українських реаліях автоматизація тестових процесів набуває особливого значення у зв'язку з необхідністю швидкого розгортання цифрових сервісів, обмеженістю ресурсів і підвищеними вимогами до безперервності роботи інформаційних систем. У цьому контексті дослідження сучасних методологій та підходів до автоматизованого тестування програмного забезпечення є важливим науково-практичним завданням.

Метою дослідження є аналіз та узагальнення сучасних методологій і підходів до автоматизованого тестування програмного забезпечення, а також визначення їх ролі у підвищенні якості та надійності програмних модулів інформаційних систем в умовах сучасного етапу розвитку інформаційних технологій.

Матеріали та методика досліджень. Матеріалами дослідження слугували наукові публікації вітчизняних і зарубіжних авторів, присвячені проблемам тестування та забезпечення якості програмного забезпечення, нормативні й методичні джерела з інженерії програмного забезпечення, а також відкриті матеріали професійних спільнот і технічна документація сучасних інструментів автоматизованого тестування.

У процесі дослідження застосовано комплекс загальнонаукових і спеціальних методів, зокрема: аналіз і синтез – для узагальнення теоретичних підходів до тестування програмного забезпечення; класифікацію – для систематизації видів, рівнів і методологій автоматизованого тестування; порівняльний аналіз – для оцінювання переваг і обмежень різних підходів та інструментів автоматизації; абстрагування та узагальнення – для формування висновків щодо ефективності застосування автоматизованого тестування в сучасних інформаційних системах.

Методологічну основу дослідження становлять принципи інженерії програмного забезпечення, концепції Agile та DevOps, а також сучасні підходи до безперервної інтеграції та доставки програмних продуктів. Отримані результати мають узагальнюючий характер і можуть бути використані як у наукових дослідженнях, так і в практичній діяльності фахівців з розробки та тестування програмного забезпечення.

Виклад основного матеріалу дослідження. Тестування програмного забезпечення є систематичним процесом аналізу програмного продукту та супровідної документації з метою виявлення дефектів, оцінювання відповідності заданим вимогам і підвищення загального рівня якості програмного рішення. У сучасних умовах розвитку інформаційних систем тестування розглядається як невід'ємна складова життєвого циклу програмного забезпечення, що безпосередньо впливає на надійність, безпеку та ефективність цифрових сервісів, які активно впроваджуються в Україні.

Залежно від способу виконання тестування поділяють на ручне та автоматизоване. Ручне тестування передбачає перевірку програмного забезпечення без використання спеціалізованих інструментів автоматизації та ґрунтується на діях і професійному досвіді тестувальника. До цього виду належать перевірка логіки роботи інтерфейсу, зручності користування, коректності відображення елементів і загальної поведінки системи. Автоматизоване тестування, своєю чергою, реалізується за допомогою програмних засобів, які виконують заздалегідь підготовлені тестові сценарії, що дає змогу значно підвищити швидкість, повторюваність і точність тестування [4].

Автоматизоване тестування доцільно впроваджувати в стабільних програмних продуктах або модулях, які часто використовуються та зазнають обмеженої кількості змін. За таких умов автоматизація є економічно обґрунтованою, оскільки зменшує витрати часу та людських ресурсів. Принципова відмінність між ручним і автоматизованим тестуванням полягає в рівні залежності від людського фактора: у ручному тестуванні результати значною мірою залежать від уважності та досвіду спеціаліста, тоді як автоматизовані тести можуть виконуватися багаторазово без безпосередньої участі людини та з однаковою точністю [4].

Усі види тестування програмного забезпечення доцільно умовно об'єднати в три групи [2; 4]:

1. *Функціональні види тестування* спрямовані на перевірку реалізації функцій програмного продукту та його взаємодії з іншими системами, включаючи перевірку відповідності вимогам, коректності обробки даних і базових аспектів безпеки.

2. *Нефункціональне тестування* орієнтоване на оцінювання характеристик, що піддаються вимірюванню, таких як продуктивність, масштабованість, зручність користування, надійність та відновлення після збоїв.

3. *Тестування, пов'язане зі змінами*, застосовується після модифікації коду та спрямоване на перевірку стабільності роботи системи; до нього належать регресійне тестування, перевірка базової працездатності системи та тестування готовності збірки до впровадження.

Процеси розробки та тестування програмного забезпечення є взаємопов'язаними, тому методології тестування тісно інтегруються з методологіями розробки. Під методологією тестування розуміють сукупність принципів, методів і підходів, що застосовуються під час планування, організації та виконання тестових робіт у межах проєкту.

Однією з класичних моделей організації процесу розробки та тестування є *каскадна модель (модель водоспаду)* [2]. Вона передбачає послідовне виконання етапів: формування вимог, аналіз і проєктування системи, реалізація програмного коду, тестування та приймання готового продукту. У межах цієї моделі команда тестування залучається вже на етапі формування вимог, де розробляється стратегія тестування, проте безпосереднє виконання тестів здійснюється після завершення етапу розробки.

Каскадна модель характеризується чіткою структурованістю та простотою управління, що є її основною перевагою (рис. 1). Водночас вона вимагає детального й повного визначення вимог на початковому етапі, що в умовах реальних проєктів, зокрема в українському цифровому середовищі, не завжди є можливим. Обмежена гнучкість і складність внесення змін роблять цю модель менш придатною для великих або динамічних інформаційних систем.

Ітеративна (ітераційна) модель розробки програмного забезпечення ґрунтується на поетапному створенні програмного продукту з багаторазовим повторенням циклів розробки, аналізу та вдосконалення отриманих результатів. Сутність цієї моделі полягає в тому, що програмний продукт створюється послідовними ітераціями, кожна з яких завершується аналізом досягнутих результатів і, за необхідності, коригуванням вимог, архітектурних рішень або функціональності [5].

Після завершення кожної ітерації обов'язково проводиться етап тестування, результати якого враховуються під час планування наступного циклу розробки. Такий підхід забезпечує раннє виявлення дефектів, підвищує керованість процесу створення програмного забезпечення та дає змогу більш точно прогнозувати обсяги і тривалість тестування на наступних етапах. В умовах сучасної розробки інформаційних систем в Україні ітеративна модель є особливо актуальною завдяки своїй гнучкості та здатності адаптуватися до змінних вимог (рис. 2).

У межах ітеративної моделі доцільно виділити чотири основні фази [1; 2].

Рис. 1. Узагальнена модель життєвого циклу каскадної методології розробки та тестування програмного забезпечення

Джерело: власна розробка автора

Рис. 2. Узагальнена модель життєвого циклу ітеративної методології розробки та тестування програмного забезпечення

Джерело: власна розробка автора

Фаза формування вимог є початковим етапом, під час якого здійснюється аналіз потреб замовника, цілей проєкту та вхідних даних. Результатом цієї фази є формування технічного завдання, що визначає ключові вимоги, пріоритети, порядок і строки розробки програмного продукту.

Фаза проєктування та розробки передбачає створення прототипу або концептуальної моделі майбутнього програмного рішення, після чого здійснюється безпосередня реалізація програмного коду відповідно до визначених вимог і проєктних рішень.

Фаза тестування та впровадження охоплює перевірку функціональної та нефункціональної відповідності програмного продукту, виявлення і усунення помилок, а також впровадження результатів ітерації в тестове або робоче середовище.

Фаза оцінювання є завершальним етапом ітерації, на якому здійснюється аналіз результатів розробки та тестування, оцінюється якість реалізованих рішень і приймаються рішення щодо доцільності подальших удосконалень у наступних циклах.

До основних переваг ітеративної моделі належить зниження ризиків, оскільки потенційні невідповідності між вимогами та реалізацією виявляються на ранніх етапах. Важливою перевагою є також налагодження ефективного зворотного зв'язку між учасниками проєкту та можливість швидкого отримання працездатних версій продукту з мінімальною кількістю критичних недоліків. Водночас недоліками моделі є підвищені вимоги до організації комунікації та те, що не всі типи вимог однаково добре піддаються ітеративному уточненню протягом усього життєвого циклу продукту.

Екстремальне програмування (Extreme Programming, XP) є однією з методологій гнучкої розробки програмного забезпечення, що передбачає короткі цикли розробки та активну взаємодію з замовником. Проєкт розбивається на невеликі, чітко визначені інженерні завдання, програмісти реалізують прості функціональні модулі та регулярно отримують зворотний зв'язок від замовника. На основі отриманих коментарів відбувається коригування вимог та подальша розробка [3].

Особливістю XP є парне програмування (pair programming) – методика, за якою весь код створюється двома програмістами на одному робочому місці. Це забезпечує кращу якість коду, підвищує його зрозумілість та полегшує підтримку. XP застосовується переважно у проєктах із високою динамікою змін вимог, де важливо швидко адаптувати програмне рішення до нових умов [6].

У XP тестування інтегровано безпосередньо у процес розробки. Основним інструментом є модульні тести (unit tests), які створюють самі розробники. Перший принцип полягає в тому, що кожен модуль має супроводжуватися модульним тестом, що забезпечує автоматичне регресійне тестування та своєчасне виявлення більшості помилок під час програмування. Другий принцип XP – тест визначає код: частина коду потрапляє до репозиторію лише після успішного проходження тестів.

До ключових переваг XP належать [4]: неперервна інтеграція – нові модулі одразу інтегруються в репозиторій, що дозволяє виявляти конфлікти на ранніх етапах та простота коду – завдяки парному програмуванню код залишається зрозумілим і легко підтримується.

Серед недоліків методології слід зазначити необхідність постійного залучення замовника, без чого ефективність розробки знижується. Крім того, парне програмування вимагає висококваліфікованих спеціалістів, яких не завжди легко знайти. Методологія XP найкраще застосовується для розробки програмного забезпечення з високою частотою змін вимог та потребою швидкої адаптації.

На рис. 3 представлено схему життєвого циклу Agile-методології, що демонструє інтеграцію розробки та тестування у рамках ітераційного процесу.

Рис. 3. Узагальнена модель екстремального програмування (Extreme Programming, XP)

Джерело: власна розробка автора

Agile-методологія є гнучким підходом до управління проєктами, що широко застосовується в малих і середніх командах розробників. Основою Agile є поділ проєкту на ітерації, постійне співробітництво між учасниками та безперервне вдосконалення продукту. Команди дотримуються циклу планування → виконання → оцінка, що дозволяє оперативно реагувати на зміни та оптимізувати процеси розробки [10].

Найпоширеніші підходи Agile – Scrum та Kanban. Основна відмінність Agile від традиційних методологій полягає в активній взаємодії багатофункціональних команд, відкритому спілкуванні, спільній роботі та довірі між учасниками процесу. Agile дозволяє інтегрувати тестування на всіх етапах розробки, що підвищує якість кінцевого продукту та скорочує час на виявлення помилок [7].

Методологія DevOps (Development and Operations) є сучасним підходом до організації процесів розробки та експлуатації програмного забезпечення, спрямованим на скорочення життєвого циклу проєктів та забезпечення безперервної поставки високоякісного програмного забезпечення (ПЗ) [9]. Основна ідея полягає у тісній інтеграції роботи розробників та фахівців з експлуатації інформаційних систем, що дозволяє ефективно координувати процеси створення та підтримки продукту.

Ключовими принципами DevOps є [6; 9]: *регулярна взаємодія та комунікація* – команда працює скоординовано, враховує потреби кожного члена та забезпечує своєчасне реагування на зміни; *неперервне розгортання* – поступове впровадження змін дозволяє оперативно доставляти продукт і при необхідності швидко скасовувати оновлення у разі виявлення помилок; *загальна відповідальність* – усі члени команди несуть відповідальність за результат, що стимулює колективну увагу до якості продукту; *виявлення та усунення проблем на ранніх етапах* – DevOps передбачає швидке виконання завдань у життєвому циклі, що дозволяє зменшити витрати часу на виправлення дефектів.

Важливим аспектом DevOps є неперервне тестування та удосконалення продукту. Процес працює наступним чином: після розгортання тестового середовища автоматично виконуються модульні тести (unit tests), які виявляють дефекти ще на ранньому етапі. Після успішного проходження модульних тестів продукт переходить на етап функціонального тестування, яке виконує спеціаліст із тестування. Такий підхід дозволяє зменшити тимчасові витрати на виявлення та усунення помилок, підвищити стабільність та надійність програмного забезпечення [9]. На рисунку 4 представлено схему життєвого циклу DevOps методології.

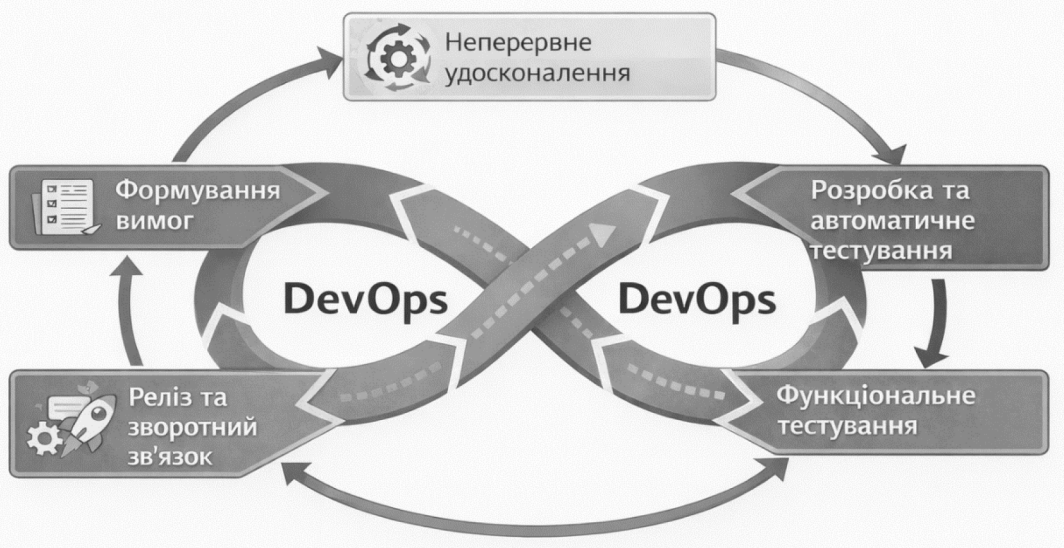

Рис. 4. Життєвий цикл DevOps методології

Джерело: власна розробка автора

До переваг методології DevOps належать [6; 8]:

- підвищення якості продукту за рахунок автоматизації процесів та швидкого виявлення дефектів;
- скорочення часу розробки та підвищення конкурентоспроможності продукту;
- колективна відповідальність за кінцевий результат, що сприяє більшій дисципліні та контролю на всіх етапах проєкту.

Серед недоліків слід зазначити необхідність високого рівня кваліфікації учасників команди, значні початкові витрати на автоматизацію процесів та потребу в ефективній організації комунікацій між усіма членами проєктної групи.

DevOps є особливо актуальною для українських IT-компаній, що працюють над проєктами з високою швидкістю змін вимог, інтеграцією з хмарними сервісами та потребою швидкого та надійного релізу програмних продуктів.

Швидкий розвиток інформаційних технологій спричиняє постійне створення нових програмних продуктів. Щоб програмне забезпечення залишалося конкурентоспроможним та економічно вигідним, необхідно забезпечувати його якість і безперервно контролювати його роботу. Тестування є невід'ємним етапом процесу розробки, а в умовах, коли системи постійно розширюються, оновлюються та активно експлуатуються, автоматизація тестування стає ключовим інструментом забезпечення якості та моніторингу стану програмних продуктів [7].

Для ефективного впровадження автоматизованого тестування важливо визначити, які саме елементи процесу підлягають автоматизації. Сучасні підходи до автоматизації тестування передбачають багаторівневу організацію тестів, що дозволяє оптимізувати ресурси команди розробників, скоротити час на виявлення дефектів та забезпечити стабільну якість продукту на всіх етапах життєвого циклу.

У літературі та практичних дослідженнях часто виділяють п'яту модель стратегічного розподілу автоматизованих тестів, відому як піраміда автоматизації тестів [13, с. 112–115]. Піраміда автоматизації тестів ілюструє оптимальне співвідношення між різними типами тестів (рис. 5) [9, с. 78–84]:

1. *Unit-тести (модульне тестування)* – розташовані в основі піраміди, їх має бути найбільше. Unit-тести забезпечують перевірку окремих функціональних блоків та компонентів системи, швидко виконуються і дозволяють раннє виявлення дефектів.
2. *Інтеграційні тести* – розташовані у середньому шарі піраміди. Вони перевіряють взаємодію між компонентами системи та дозволяють оцінити сумісність модулів та коректність обміну даними.
3. *Системні та енд-ту-енд тести* – розташовані на вершині піраміди, їх кількість мінімальна через високу вартість виконання. Вони забезпечують перевірку повної функціональності та відповідності системи вимогам замовника.

Рис. 5. Піраміда автоматизації тестів

Джерело: власна розробка автора

Такий підхід дозволяє досягти балансу між обсягом тестування та витратами на його виконання, сприяє ефективній автоматизації та підтримує стабільність продукту на всіх етапах розробки. Піраміда автоматизації тестів також слугує стратегічним інструментом при плануванні тестової діяльності, оскільки забезпечує пріоритетність модульних тестів і оптимізує ресурси для інтеграційного та системного тестування.

Перед початком автоматизації тестування необхідно визначитися з технологіями, побудувати план та проаналізувати існуючі підходи. Основне питання, яке слід вирішити: що саме підлягає автоматизації та яку мету вона переслідує? Відповідь на це питання визначає ефективність процесу та економічну доцільність автоматизації. Серед ключових рекомендацій для автоматизації виділяють [9]: тести, що виконуються при кожній збірці та релізі (регресійні, приймальні); тести з однаковим робочим процесом, але різними вхідними даними (аналітика граничних значень); тести, які потребують збору різної інформації (SQL-запити, низькорівневі атрибути сервісів); тести з великими обсягами даних; тести, що потребують багато часу або обмежені часовими рамками; тести на різних конфігураціях (операційні системи, версії збірки); тести, що потребують документування через скріншоти для доказової бази.

Перед проведенням детального аналізу технологій автоматизованого тестування було здійснено класифікацію існуючих підходів за критеріями повторного використання, складності впровадження та масштабу застосування. Визначено, що кожна методика має свої переваги та недоліки, які необхідно враховувати при побудові стратегії автоматизації тестів.

У таблиці 1 представлені ключові методи автоматизації тестування програмного забезпечення, їхні переваги та обмеження, а також приклади практичного застосування. Таке узагальнення дозволяє систематизувати підходи та забезпечити науково обґрунтований вибір технологій у конкретному проєкті.

Таблиця 1. Переваги та недоліки технологій автоматизованого тестування

Метод автоматизації	Переваги	Недоліки	Джерела/Приклади
Часткові рішення (Ad hoc scripts)	Швидка реалізація, використання будь-яких інструментів, простота написання коду	Відсутність універсальності, низька повторюваність, складність підтримки	Python, Bash, SQL
Data-Driven Testing (DDT)	Повторне використання тестів, зменшення надлишковості сценаріїв, зручне зберігання даних	Складна реалізація, ризик конфліктів між тестами, потребує рефакторингу при зміні логіки	CSV, Excel, бази даних
Keyword-Driven Testing (KDT)	Тестування UI без знання програмування, повторне використання ключових слів, розширене управління тестами	Високі ресурси на впровадження, потребує кваліфікованих співробітників, складне планування	Selenium IDE, Robot Framework
Record & Playback (R&P)	Простота створення автотестів, автоматизація рутинних дій, швидке прототипування	Лінійність сценаріїв, обмежена гнучкість, низька надійність, складність масштабування	Katalon Studio, Selenium IDE
Behaviour Driven Development (BDD)	Зрозумілі сценарії для бізнесу та розробників, інтеграція DevOps та CI/CD, спрощене створення тестів	Потребує дисциплінованого планування, узгоджені практики, додаткові ресурси	Cucumber, Rest-Assured, Gherkin

1. Часткові рішення (Ad hoc scripts). Даний підхід показує, що для кожного завдання створюється окремий скрипт на будь-якій високорівневій мові програмування. Наприклад, автоматизація наповнення бази даних тестовими даними:

```
import sqlite3

conn = sqlite3.connect('test_db.sqlite')
cursor = conn.cursor()
cursor.execute(«INSERT INTO users (name, email) VALUES (?, ?)», ('Test User', 'test@
example.com'))
conn.commit()
conn.close()
```

2. Data-Driven Testing (DDT). Дозволяє виконувати один і той же тест з різними наборами вхідних даних, що зберігаються поза тестом, наприклад у CSV:

```
import csv
import unittest

class TestLogin(unittest.TestCase):
  def test_login(self):
    with open('login_data.csv') as f:
      reader = csv.reader(f)
      for row in reader:
        username, password = row
        self.assertTrue(authenticate(username, password))
```

3. Keyword-Driven Testing (KDT). Розділяє створення тестів та їх виконання за ключовими словами, які описують дії користувача:

```
def execute_keyword(keyword, obj, value=None):
  if keyword == 'Click':
    click(obj)
  elif keyword == 'Input':
    input_text(obj, value)
```

4. Record & Playback. Тестові сценарії записуються на основі дій користувача. Приклад у Katalon Studio або Selenium IDE.

5. Behaviour Driven Development (BDD). Фокус на очікувану поведінку системи, зрозумілу для бізнесу і розробників. Приклад у Cucumber:

```
Feature: Login
  Scenario: Successful login
    Given user navigates to login page
    When user enters valid credentials
    Then user is redirected to dashboard
```

Проведений аналіз показує, що жодна технологія автоматизації не є універсальною. Оптимальна стратегія передбачає комбінування підходів:

- Unit-тести та DDT – забезпечують базову автоматизацію та надійність.
- KDT та BDD – дозволяють автоматизувати UI та поведінкові сценарії.
- Record & Playback – підходить для швидкого прототипування автотестів, але не для великомасштабних проєктів.

Комбіноване використання цих технологій дозволяє зменшити час на тестування, підвищити стабільність системи та забезпечити економічну доцільність автоматизації, що особливо актуально для розробки сучасного програмного забезпечення в Україні.

Висновки. У ході дослідження встановлено, що тестування програмного забезпечення є обов'язковою складовою процесу розробки інформаційних систем і суттєво впливає на рівень їх якості, надійності та стабільності. В умовах зростання складності програмних продуктів і динамічності вимог автоматизоване тестування розглядається як ефективний інструмент підвищення керованості та повторюваності тестових процесів.

Узагальнення наукових джерел дало змогу систематизувати основні види та рівні тестування програмного забезпечення, а також визначити їх роль у забезпеченні функціональної та нефункціональної відповідності програмних рішень. Розглянуті методології розробки і тестування демонструють тісний взаємозв'язок між процесами створення програмного коду та контролю його якості протягом усього життєвого циклу програмного забезпечення.

Аналіз підходів до автоматизації тестування підтверджує доцільність багаторівневої організації тестів та раціонального поєднання різних технологій автоматизації залежно від завдань проєкту. Використання структурованих моделей розподілу тестів сприяє оптимізації ресурсів і підвищенню ефективності тестових робіт. Отримані результати мають узагальнюючий характер і можуть бути використані як теоретична основа для подальших досліджень у сфері тестування програмного забезпечення, а також як методичні орієнтири при плануванні та організації тестових процесів у сучасних інформаційних системах.

Перспективи подальших досліджень полягають у розробці адаптивних моделей автоматизованого тестування з використанням методів штучного інтелекту та машинного навчання для підвищення ефективності виявлення дефектів. Доцільним є також дослідження інтеграції інструментів тестування з DevOps-практиками та CI/CD-середовищами з метою забезпечення безперервного контролю якості програмних продуктів. Окремої уваги потребує оцінювання економічної ефективності впровадження автоматизованого тестування в проєктах різного масштабу та складності.

Список використаних джерел

1. Бородіна О. О., Педченко О. І. Автоматизація тестування програмного забезпечення засобами фреймворків. *Математичне та імітаційне моделювання систем. МОДС 2018*: тези доповідей Тринадцятої міжнародної науково-практичної конференції (Чернігів, 25–29 червня 2018 р.) / М-во освіти і науки України, Нац. Акад. наук України, Академія технологічних наук України, Інженерна академія України та ін. Чернігів: ЧНТУ, 2018. С. 343–345. URL: https://reposit.nupp.edu.ua/handle/PoltNTU/4364.
2. Hazin K. S., Volokyta A. M. Video game test automation approach. *Міжвідомчий науково-технічний збірник «Адаптивні системи автоматичного управління»*. 2025. 2(47), 45–52. https://asac.kpi.ua/article/view/340165/328212
3. Киричек Г. М., Тягунова М. М., Курач А. В. Автоматизоване тестування веб-платформ з використанням Java та Selenium. *Інформаційні технології та суспільство*. 2021. № 3. С. 87–93. DOI: https://doi.org/ 10.32689/MAUP.IT.2022.1.4
4. Максимович М. В., Мичуда Л. З. Роль, проблеми та методи автоматизації тестування безпеки програмного забезпечення. *Computer systems and networks*. 2024. Vol. 6, No. 2. С. 131–140. DOI: https://doi.org/10.23939/csn2024.02.131
5. Мартинова О. В., Богач І. В. Методи тестування програмного забезпечення: автоматизація та її переваги. *Вісник Вінницького національного технічного університету*. 2020. № 4. С. 98–103. URL: https://ir.lib.vntu.edu.ua/handle/123456789/44265
6. Суприган В. Дослідження стратегії багатоетапного розгортання програмного забезпечення з виключенням сторонніх ефектів. *Вісник Хмельницького національного університету*. Серія: технічні науки. 2024. № 2 (333). С. 125–150. DOI: 10.31891/2307-5732-2024-333-2-19
7. Цирульник С. М., Бородай Я. О., Ткачук В. М., Непийвода М. В. Програмне резервування мікропроцесорних систем. *Scientific Collection «InterConf», (49)*: with the Proceedings of the 2 nd International Scientific and Practical Conference «Theory and Practice of Science: Key Aspects» (April 7- 8, 2021). Rome, Italy: Dana, 2021. P. 611–621. DOI: https://doi.org/10.51582/interconf.7-8.04.2021.067
8. Ушакова І. О. Підходи до забезпечення якості програмного забезпечення. Сучасні інформаційні технології і системи: монографія / за заг. ред. В. С. Пономаренка. Харків: «Стильіздат», 2021. С. 125-140.
9. Bass L., Weber I., Zhu L. DevOps: A Software Architect's Perspective. Boston : Addison-Wesley Professional, 2015. 368 p.
10. Brynza N., Lobanov K. Analysis of the effectiveness of software testing technologies for information systems. *SWorldJournal*. 2023. С. 11-22. DOI: https://doi.org/10.30888/2663-5712.2026-35-01-043
11. Martin R. C. *Clean Agile: Back to Basics*. Pearson / Addison-Wesley, 2019. 240 p.
12. Meszaros G. *xUnit Test Patterns: Refactoring Test Code*. Addison-Wesley Professional, 2007. 912 p.
13. Riccio V., Jahangirova G., Stocco A., Humbatova N., Weiss M., Tonella P. Testing machine learning based systems: a systematic mapping. *Empirical Software Engineering*, 2020. 25. С. 5193–5254. DOI: https://doi.org/10.1007/s10664-020-09881-0

Lebid O. V.
Doctor of Philosophy in Economics,
Senior Lecturer at the Department of Computer Science and Digital Economy,
Vinnytsia National Agrarian University
Vinnytsia, Ukraine
E-mail: *sshlebid@gmail.com*
ORCID: *0000-0003-4253-8696*

AUTOMATION OF SOFTWARE TESTING: KEY METHODOLOGIES AND APPROACHES

Abstract

The development of software testing constitutes an important component of modern software systems engineering, with more than seventy years of formation and continuous improvement. Automated testing, as a scientific and applied discipline, today plays a key role in enhancing the quality, reliability, and resilience of software products across various application domains. This is particularly relevant for national information systems of Ukraine, which operate under conditions of digital transformation, integration with European standards, and an increased demand for secure and fault-tolerant solutions supporting public, economic, and social processes.

The article provides a systematic analysis and classification of the main types of software testing, including functional, non-functional, unit, integration, system, and regression testing. A review of modern approaches to test automation is conducted, based on current methodologies-specifically Agile and DevOps practices, as well as the shift-left and shift-right concepts, which enable early defect detection and continuous quality assessment of software modules.

Special attention is given to the description of levels of automated testing and the technological tools applied in the study, including framework-based tools for automated test generation, continuous integration and delivery (CI/CD) environments, and practical integration with version control systems. The advantages and limitations of automated testing are analyzed in the context of Ukrainian software development projects, with emphasis on the need to adapt testing strategies to constrained resources, dynamic changes in requirements, and the necessity for rapid implementation of updates.

The research results substantiate that the implementation of automated testing contributes to improving the efficiency of engineering processes, minimizing the number of errors in production environments, and strengthening trust in software products among both internal users and international partners.

Key words: *automated testing, software, software quality, information systems, unit testing, testing methodologies, test process automation.*

References

1. Borodina, O. O., & Pedchenko, O. I. (2018). Avtomatyzatsiia testuvannia prohramnoho zabezpechennia zasobamy freimvorkiv [Automation of software testing using frameworks]. *Matematychne ta imitatsiine modeliuvannia system. MODS 2018* : tezy dopovidei Trynadtsiatoi mizhnarodnoi naukovo-praktychnoi konferentsii (Chernihiv, 25–29 chervnia 2018 r.) / M-vo osvity i nauky Ukrainy, Nats. Akad. nauk Ukrainy, Akademiia tekhnolohichnykh nauk Ukrainy, Inzhenerna akademiia Ukrainy ta in. Chernihiv : ChNTU (pp. 343–345). https://reposit.nupp.edu.ua/handle/PoltNTU/4364 [in Ukrainian].

2. Hazin, K. S., & Volokyta, A. M. (2025). Video game test automation approach. *Mizhvidomchyi naukovo-tekhnichnyi zbirnyk "Adaptyvni systemy avtomatychnoho upravlinnia"*, 2(47), 32–40. https://asac.kpi.ua/article/view/340165/328212 [in English].

3. Kyrychek, H. M., Tiahunova, M. M., & Kurach, A. V. (2021). Avtomatyzovane testuvannia veb-platform z vykorystanniam Java ta Selenium [Automated testing of web platforms using Java and Selenium]. *Informatsiini tekhnolohii ta suspilstvo – Information Technologies and Society*, 3, 87–93. DOI: https://doi.org/ 10.32689/MAUP.IT.2022.1.4 [in Ukrainian].

4. Maksymovych, M. V., & Mychuda, L. Z. (2024). Rol, problemy ta metody avtomatyzatsii testuvannia bezpeky prohramnoho zabezpechennia [The role, problems and methods of automation of software security testing]. *Computer Systems and Networks*, 6(2), 131–140. DOI: https://doi.org/10.23939/csn2024.02.131 [in Ukrainian].

5. Martynova, O. V., & Bohach, I. V. (2020). Metody testuvannia prohramnoho zabezpechennia: avtomatyzatsiia ta yii perevahy [Software testing methods: automation and its advantages]. *Visnyk Vinnytskoho natsionalnoho tekhnichnoho universytetu – Bulletin of Vinnytsia National Technical University*, 4, 98–103. https://ir.lib.vntu.edu.ua/handle/123456789/44265 [in Ukrainian].

6. Supryhan, V. (2024). Doslidzhennia stratehii bahatoetapnoho rozhortannia prohramnoho zabezpechennia z vykliuchenniam storonnikh efektiv [Study of a multi-stage software deployment strategy excluding side effects]. *Visnyk Khmelnytskoho natsionalnoho universytetu. Seriia: Tekhnichni nauky – Bulletin of Khmelnytskyi National University. Technical Sciences Series*, 2(333), 125–150. https://doi.org/10.31891/2307-5732-2024-333-2-19 [in Ukrainian].

7. Tsyrulnyk, S. M., Borodai, Ya. O., Tkachuk, V. M., & Nepyivoda, M. V. (2021). Prohramne rezervuvannia mikroprotsesornykh system [Software redundancy of microprocessor systems]. *Scientific Collection "InterConf", (49)*: with the Proceedings of the 2nd International Scientific and Practical Conference "Theory and Practice of Science: Key Aspects" (April 7–8, 2021). Rome, Italy: Dana, 611–621. https://doi.org/10.51582/interconf.7-8.04.2021.067 [in Ukrainian].

8. Ushakova, I. O. (2021). Pidkhody do zabezpechennia yakosti prohramnoho zabezpechennia [Approaches to software quality assurance]. In: V. S. Ponomarenko (Ed.), *Suchasni informatsiini tekhnolohii i systemy* (pp. 125–140). Kharkiv : Stylizdat [in Ukrainian].

9. Bass, L., Weber, I., & Zhu, L. (2015). DevOps: *A Software Architect's Perspective*. Boston : Addison-Wesley Professional, 368 p. [in English].

10. Brynza, N., & Lobanov, K. (2023). Analysis of the main problems of automated software testing. In: *Analysis of the effectiveness of software testing technologies for information systems*. *SWorldJournal*, 11–22. https://doi.org/10.30888/2663-5712.2026-35-01-043 [in English].

11. Martin, R. C. (2019). Chystyi Agile: Povernennia do osnov [Clean Agile: Back to Basics]. *Pearson / Addison-Wesley*, 240 p. [in English].

12. Meszaros, G. (2007). Shablony modulnykh testiv: refaktorynh testovoho kodu [xUnit Test Patterns: Refactoring Test Code]. *Addison-Wesley Professional*, 912 p. [in English].

13. Riccio, V., Jahangirova, G., Stocco, A., Humbatova, N., Weiss, M., & Tonella, P. (2020). Testing machine learning based systems: a systematic mapping. *Empirical Software Engineering*, 25, 5193–5254. https://doi.org/10.1007/s10664-020-09881-0 [in English].

Дата першого надходження статті до видання: 16.01.2026
Дата прийняття статті до друку після рецензування: 04.03.2026
Дата публікації (оприлюднення) статті: 27.04.2026