**УДК 004.05**
**DOI https://doi.org/10.32782/tnv-tech.2023.2.3**

# ОСОБЛИВОСТІ РОЗРОБКИ ТА ВИКОРИСТАННЯ ФРЕЙМВОРКІВ ДЛЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ

***Гакман Д. В.*** *– магістр*
*Державного університету телекомунікацій*
*ORCID ID: 0009-0007-5952-0667*

***Антоненко А. В.*** *– кандидат технічних наук, доцент,*
*доцент кафедри комп'ютерної інженерії*
*Державного університету телекомунікацій*
*ORCID ID: 0000-0001-9397-1209*

*Для успішного виходу на ринок, а також стабілізації та утримання своїх позицій на ньому, кожен продукт проходить тестування в різних його проявах та на різних етапах свого життєвого циклу. Це важлива частина процесу розробки та підтримки, адже чим далі розвивається IT індустрія, тим більше з'являється вимог до кінцевого продукту, щоб він був конкурентоспроможним. Так відбувається тому, що важливі характеристики такі як стабільність, зручність у використанні чи продуманий дизайн інтерфейсу взаємодії з користувачем стали нормою. В сучасному світі, щоб постійно отримувати нових клієнтів – потрібно принести більше ніж це. Кожна характеристика чи вимога повинні перевірятись ще детальніше. За різними оцінками загальна вартість тестування програмного забезпечення може становити від 15 до 25 % від загальної вартості проекту. А тому до нього потрібно підходити з продуманим планом та підготовкою. Тут продукти можна умовно розділити на два напрями: що після виходу на ринок вже не будуть допрацьовуватись і що будуть працювати над покращеннями впродовж всього свого циклу існування. В першому випадку часто достатньо провести повне мануальне тестування та в разі успіху випустити продукт. В другому ж випадку тестувати доведеться регулярно, при кожних змінах чи оновленні. А отже, це буде забирати значну частину ресурсів та бюджету. Тому на таких проектах часто присутня автоматизація тестування, яка суттєво зменшує кількість ручної роботи. Вона часто може обходитись дешевше, а також дозволяє значно зменшити витрати часу та ресурсів та забезпечує більшу точність та ефективність в порівнянні з ручним тестуванням, яке часто повторюється. Однак розробляти нову систему для запуску та розробки тест кейсів для кожного проекту – це дуже складно і витратно. Тому спеціалісти розробили універсальні фреймворки для автоматизованого тестування, що гнучко налаштовуються під індивідуальні потреби. Використання фреймворків має свої виклики, які можуть вплинути на ефективність тестування. У даній статті розглянуто три найбільш поширені проблеми, з якими стикаються автоматизатори, та способи їх вирішення.*

***Ключові слова:*** *тестовий фреймворк, тестування, автоматизація, тест кейс, інструменти для автоматизації.*

***Hakman D. V., Antonenko A. V. Features of the development and use of frameworks for automated testing***

*In order to successfully enter the market, as well as to stabilize and maintain its position on it, each product undergoes testing in its various manifestations and at various stages of its life cycle. This is an important part of the development and support process, because the further the IT industry develops, the more requirements appear for the final product to be competitive. This is because important features such as stability, ease of use, or thoughtful user interface design have become the norm. In today's world, to constantly get new customers, you need to bring more than this. Each characteristic or requirement should be checked in more detail. According to various estimates, the total cost of software testing can be from 15 to 25 % of the total cost of the project. And therefore it is necessary to approach it with a well-thought-out plan and preparation. Here, products can be conditionally divided into two directions: those that will not be improved after entering the market and that will work on improvements throughout their entire life cycle. In the first case, it is often enough to conduct full manual testing and release the product if successful. In the second case, you will have to test regularly, with every change or update. And therefore, it will take a significant part of resources and budget. Therefore, such projects often have testing*

*automation, which significantly reduces the amount of manual work. It can often be cheaper, and can significantly reduce time and resources, and provide greater accuracy and efficiency compared to manual testing, which is often repeated. However, developing a new system for launching and developing test cases for each project is very difficult and expensive. Therefore, specialists have developed universal frameworks for automated testing that are flexibly adjusted to individual needs. Using frameworks has its own challenges that can affect test performance. This article examines three of the most common problems faced by automatizers and how to solve them.*

***Key words:*** *test framework, testing, automation, test case, automation tools.*

**Вступ.** В сучасному світі, де швидкість розробки програмного забезпечення має ключове значення, тестування ПЗ, безсумнівно, є невід'ємною частиною життєвого циклу розробки (SDLC). З розвитком Agile та DevOps методологій, а також прагненням підприємств до швидких релізів та якісних продуктів, виникла потреба в методах тестування програмного забезпечення, які є швидшими та ефективнішими, ніж ручне тестування. Саме в цей момент архітектура автоматизації тестування програмного забезпечення, що реалізується за допомогою фреймворків, набуває все більшого поширення і сьогодні лідирує в тестуванні ПЗ [1–3].

Test Automation Framework – це не окремий інструмент чи процес, а їх набір, який спільно працює для підтримки безпосередньо автоматизованого тестування, розробки тестових випадків, генерації звітів чи інтеграції із іншими інструментами. Він об'єднує різні частини, такі як бібліотеки, тестові дані та різноманітні багаторазові модулі. Це концептуальна частина автоматизованого тестування, яка допомагає тестувальникам більш ефективно використовувати ресурси. Фреймворк можна визначити як набір правил або найкращих практик, яких можна дотримуватися систематично, що гарантує отримання бажаних результатів. В загальному це платформа, розроблена шляхом інтеграції різних апаратних і програмних ресурсів, а також використання різних інструментів на основі кваліфікованого відбору. Він дозволяє ефективно проєктувати і розробляти сценарії автоматизованого тестування, а також забезпечує надійний аналіз проблем або помилок для об'єкту, що тестується [4–6].

При розробці та використанні фреймворків, автоматизатори зіткнуться з рядом проблем, що можуть вплинути на якість тестування та витрати на нього. Певна частина цих проблем буде унікальна до проєкту, що розробляється, але багато з них будуть або однаковими, або схожими за своїм сенсом чи походженням, а, отже, буде достатньог адаптувати існуюче рішення, замість розробки нового. Тому, щоби принести найбільшу користь та мати змогу правильно розподіляти ресурси проєкту важливо розуміти ті проблеми, які часто зустрічаються і вміти коригувати існуючі рішення під власні потреби, адже фреймворки залишаються важливим інструментом для забезпечення якості програмного забезпечення. Використовуючи правильний підхід можна суттєво знизити витрати на тестування та скоротити час, необхідний для випуску нових версій ПЗ. Нижче будуть розглянуті основні складнощі, що часто зустрічаються при розробці та використанні фреймворків для автоматизованого тестування, а також запропоновані декілька можливих шляхів вирішення цих проблем із окремим розбором кожного з них. Це дасть змогу поглибити своє розуміння, а також мати в арсеналі декілька варіантів підходів, які можна комбінувати для досягнення найбільш вигідного результату.

**Постановка проблеми.** Одним з основних викликів в автоматизації – є необхідність забезпечити покриття тестами різноманітних вимог, як функціональних, так і нефункціональних, при цьому підтримуючи високу швидкість виконання

тестів і максимальну автономність від людини. Тому це вимагає більшої кількості знань та вмінь від інженерів-автоматизаторів, оскільки неправильне використання та налаштування фреймворку, може призвести до погіршення процесу тестування, що матиме негативний вплив на весь проєкт [7–10].

Проблеми, що розглядаються в даній статті мають важливий практичний зв'язок із завданнями в галузі розробки програмного забезпечення та тестування. Адже перед розробниками тестових фреймворків завжди стоїть завдання, якнайкраще забезпечити підтримку та перевірку якості продукту

**Мета дослідження.** *Метою роботи* є поглиблення розуміння проблем, пов'язаних з використанням фреймворків для автоматизованого тестування, та надання інформації про можливі шляхи їх вирішення.

Об'єктом дослідження є фреймворки для автоматизованого тестування.

Предметом дослідження є проблеми, пов'язані з використанням фреймворків для автоматизованого тестування та способи їх вирішення.

**Аналіз останніх досліджень і публікацій.** Аналіз інформаційних джерел, що стосуються проблем розробки та використання фреймворків для автоматизованого тестування, підтверджує важливість цієї теми та допомагає глибше розуміти наявні питання.

У науковій праці "Challenges in Test Automation Framework Design and Development" у журналі International Journal of Advanced Research in Computer Science and Software Engineering [5] досліджує проблеми, що виникають при розробці та використанні фреймворків для автоматизованого тестування. У статті описано проблеми з використанням даних, проблеми з управлінням тестовими сценаріями та скриптами, а також проблеми зі змінністю тестового середовища.

Дослідження "Issues and Challenges of Test Automation: A Systematic Literature Review" [6] розглядає проблеми, які пов'язані з автоматизованим тестуванням в цілому. Дослідження підтвердило, що багато компаній використовують фреймворки для автоматизованого тестування та часто стикаються з проблеми в їх розробці та підтримці. Дослідження зазначає, що відсутність стандартів та норм у розробці тестових сценаріїв є однією з основних проблем.

У науковій праці "The challenges and benefits of continuous integration in software engineering" у журналі Information and Software Technology [11] розглядає проблеми, пов'язані зі впровадженням Continuous Integration в процес розробки програмного забезпечення. У статті зазначено, що інтеграція тестових фреймворків є однією з найбільш складних задач при впровадженні Continuous Integration. У статті також описано переваги Continuous Integration, такі як зменшення ризиків при релізі програмного забезпечення та зменшення часу на виявлення та виправлення помилок.

Дослідження "Automation testing challenges and solutions: A review" у журналі International Journal of Computer Applications [12] досліджує складнощі, пов'язані з автоматизованим тестуванням в цілому. У статті описано проблеми, які можуть виникнути саме при використанні фреймворків, такі як складність відлагодження тестових скриптів та проблеми зі змінністю тестового середовища. Також у статті наведено різні рішення для вирішення цих проблем.

Проаналізувавши джерела, для розгляду було обрано питання, які найчастіше мають вплив на процес тестування:

1. Створення та підтримка тестових скриптів.
2. Відсутність загальноприйнятих стандартів та норм для розробки тестових сценаріїв.

3. Інтеграція тестових фреймворків в процес Continuous Integration / Countinuos Development.

Ці проблеми призводять до помилок та затримок у процесі розробки програмного забезпечення, тому їх вивчення та вирішення є важливою темою для тих, хто бере участь в розробці та тестуванні програмного забезпечення. Знання про ці проблеми допоможе розробити ефективну стратегію тестування та вибрати інструменти, які найбільш підходять для конкретного проєкту.

**Виклад основного матеріалу дослідження.** Перша проблема, з якою стикаються автоматизатори – це створення та підтримка тестових скриптів. Це завдання займає багато часу та зусиль автоматизатора, оскільки система, яку тестують, часто змінюється та доповнюється, а отже, раніше написані тести вже можуть бути неактуальними та потребувати оновлення. Коли таких тестів тисячі – це нелегке завдання, тому щоб не втрачати час та ресурси проєкту, важливо розуміти проблему та підходи до вирішення. Розберемо можливі розв'язання.

Першим способом є використання шаблонів проєктування та рефакторингу коду. Умовно можна виділити три рівні повторного використання частин коду. Нижній – це збірки класів, бібліотек, модулів. На найвищому рівні знаходяться фреймворки, адже у них важливою є тільки архітектура. Зазвичай фреймворк має набагато більший обсяг, ніж один клас. Він надає змогу задати потрібну поведінку, а потім, при виконанні певних умов, сам викликає її. До прикладу JUnit звертається до вашого класу, коли потрібно виконати тест. Все інше відбувається всередині фреймворка. На середньому ж рівні можна розмістити патерни, які є більш абстрактними ніж фреймворки, та водночас мають менше прив'язки до мови програмування. Вони є описом того, як певні класи чи методи взаємодіють один з одним. Основні переваги використання шаблонів полягають в наступному:

1. Економлять час і зусилля
2. Знижують витрати на обслуговування
3. Покращують повторне використанні коду
4. Підвищують надійність
5. Допомагають створювати структурований код, який полегшує процес автоматизації
6. Покращують комунікацію і розуміння між інженерами.

Вони допомагають відмежовувати внутрішню реалізацію фреймворку від реалізації тестів або розв'язувати проблему стандартизації коду та дозволяють використовувати рішення, що вже перевірені величезною кількістю інженерів, адаптуючи їх під власну потребу.

Шаблонами що часто використовуються в автоматизації є Page Object Model, Factory і Singleton.

Page Object Model (POM) дозволяє відокремити структуру веб-сторінки від тестового скрипту, що забезпечує більшу стабільність та зручність підтримки тест кейсів. За допомогою POM, структура веб-сторінки відображається у вигляді об'єктів, методи яких абстрагують запити та налаштування, даючи назовні тільки зручний читабельний інтерфейс. Ще однією перевагою є можливість повторного використання даних об'єктів. Якщо структура веб-сторінки змінюється, то потрібно змінити лише реалізацію об'єкта, а тест кейси залишаться незмінними.

У фабричному патерні проєктування існує клас з фабричним методом, який займається усіма процесами створення об'єктів. У цьому патерні є суперклас з декількома підкласами, і на основі даних, введених користувачем на рівні тестового класу, він повертає один з підкласів. За логіку реалізації відповідає клас, який

розширює батьківський клас, таким чином, він приховує складний код на рівні тестування. Як користувачеві, нам просто потрібно створити об'єкт цього класу і використовувати його в тесті для виклику відповідного методу, що містить бізнес-логіку.

Патерн проектування Singleton – це один з найпростіших і зрозумілих патернів, який можна впровадити у фреймворк автоматизації. Цей патерн використовується, коли потрібно використовувати один і той же об'єкт класу в різних місцях. Він обмежує можливі екземпляри класу до одного екземпляра. Щоб його імплементувати, потрібно оголосити конструктор класу закритим (private), щоб ніхто не міг створити екземпляр класу за його межами, потім оголосити статичну змінну-посилання класу і статичний метод що повертає об'єкт класу одинака. Також цей метод повинен перевіряти, чи об'єкт вже був створений один раз [13].

Наступним способом вирішення є використання інструментів, що дозволяють записувати тестові скрипти. Такі інструменти автоматично запам'ятовують дії, що виконує користувач. Прикладом є Selenium IDE що дозволяє записувати тестові дії у форматі Selenese. Це дуже зручно для автоматизаторів без достатніх знань програмування, однак, дослідження показують, що використання таких інструментів є неефективним у випадку складних тестових сценаріїв або при зміні структури веб-сторінки. Також, відсутність навичок програмування в автоматизатора може призвести до генерації неправильних тестових скриптів, тому даний підхід вимагає уваги та обов'язкового рев'ю тестів більш досвідченими колегами.

Ще одним способом є використання параметризації. При такому підході створюється один (можливо трохи складніший) тестовий скрипт, який приймає певний набір тестових варіацій, що дозволяє провести набагато більше тестів. Гарним прикладом є тестування будь-якого поля вводу. Можна створити один тестовий скрипт, який буде приймати на вхід різноманітні комбінації тексту, і перевіряти поведінку сайту в кожній із варіацій. Але варто зазначити що використання параметризації не завжди потрібно і це залежить від ситуації. Найкращим сценарієм для таких тестів є ситуація, коли у нас є багато тестових варіацій для одного тесту. З іншого боку, не варто використовувати параметризовані тести в ситуаціях, коли відбувається перевірка тільки одного набору даних. Відокремлення даних від тесту буде надмірним ускладненням.

Переваги використання параметризації:

1. Потрібен тільки один тест для багатьох тестових випадків.
2. Логіка тесту відокремлена від тестових даних.
3. Файлом з даними можна легко поділитись з іншими членами команди, а особливо це корисно, якщо потрібно поділитись із ручними тестувальниками, які не знають мови програмування.
4. Вони зменшують повторюваність коду.

Недоліки використання параметризації:

1. Відокремлення логіки тесту від параметрів тесту вимагає додаткової роботи.
2. Надмірність для тестів з невеликою кількістю тестових кейсів для перевірки.
3. Часто потрібно підтримувати додатковий файл з тестовими прикладами.

Дослідження компанії Sauce Labs, показало, що використання параметризованих тестових скриптів дозволило зменшити час, потрібний для підтримки тестових скриптів, на 70 % у порівнянні з тестовими скриптами без параметризації, а отже, у випадку змін в системі, потрібно набагато менше зусиль для виправлення тестових сценаріїв [14].

Також при розробці тестових скриптів варто керуватись 7 принципами тестування відповідно до ISTQB:

1. Тестування зменшує ймовірність того, що в програмному забезпеченні залишаться невиявлені дефекти, але навіть якщо дефектів не виявлено, тестування не є доказом правильності програми.

2. Тестування всіх комбінацій вхідних даних і передумов неможливе, за винятком тривіальних випадків. Замість того, щоб намагатися провести вичерпне тестування, слід використовувати аналіз ризиків, методи тестування та пріоритети, щоб зосередити зусилля на тестуванні.

3. Щоб виявити дефекти на ранніх стадіях, статичне і динамічне тестування слід починати якомога раніше в життєвому циклі розробки програмного забезпечення. Раннє тестування іноді називають зсувом вліво. Тестування на ранніх стадіях життєвого циклу розробки програмного забезпечення допомагає зменшити або усунути дорогі зміни.

4. Невелика кількість модулів зазвичай містить більшість дефектів, виявлених під час передрелізного тестування, або є причиною більшості операційних збоїв. Прогнозовані кластери дефектів і фактично виявлені кластери дефектів під час випробувань або експлуатації є важливим вкладом в аналіз ризиків, який використовується для фокусування зусиль під час випробувань (як зазначено в принципі 2).

5. Якщо повторювати одні й ті ж випробування знову і знову, то з часом ці випробування перестануть виявляти нові дефекти. Щоб виявити нові дефекти, може знадобитися змінити існуючі тести і дані випробувань, а також написати нові тести.

6. Тестування проводиться по-різному в різних контекстах. Наприклад, критично важливе для безпеки промислове програмне забезпечення для управління тестується інакше, ніж мобільний додаток для електронної комерції.

7. Деякі організації очікують, що тестувальники можуть провести всі можливі тести і знайти всі можливі дефекти, але принципи 2 і 1, відповідно, говорять, що це неможливо. Крім того, помилково очікувати, що просто виявлення та виправлення великої кількості дефектів забезпечить успіх системи. Наприклад, ретельне тестування всіх визначених вимог і виправлення всіх знайдених дефектів може призвести до того, що система буде складною у використанні, не відповідатиме потребам і очікуванням користувачів або буде гіршою порівняно з іншими конкуруючими системами.

Розробка тестових скриптів є трудомістким процесом, який потребує постійного удосконалення. Проте, використання правильних технологій та підходів допоможе забезпечити більш ефективне використання часу та загалом якісніше тестування програмного забезпечення.

Друга проблема, що потребує уваги – це відсутність загальноприйнятих стандартів та норм для розробки тестових сценаріїв. Часто автоматизатори можуть мати різні підходи до розробки, що призводить до складнощів у підтримці такого коду іншими QA інженерами. Не існує загальновизначених правил, яких всі повинні дотримуватися в процесі розробки тестових скриптів, а тому в кожної людини може бути своя реалізація рішення на ту чи іншу проблему. Це не завжди погано, але на початку розробки тестового фреймворку цьому точно варто приділити час.

До прикладу, дослідження “A Systematic Review of Test Automation Tools and Frameworks for Web Applications”, опубліковане в журналі IEEE Access

показало, що більшість з 50-ти проаналізованих інструментів та фреймворків для автоматизованого тестування веб-додатків, використовують власні методології тестування, що може призвести до неоднаковості та спричиняє потребу в можливому перенавчанні наявних спеціалістів [15].

Хорошим розв'язанням цього питання є розбір наявних стандартів із суміжних галузей, таких як розробка чи ручне тестування та випрацювання на їх основі стандартів для окремої команди, або групи команд, що працюють над одним проєктом. Такі стандарти можуть включати правила створення тестових скриптів, правила форматування коду для його консистентного вигляду, використання методик тестування, а також уніфікація процесів звітування та відстеження помилок. Усі ці дії допоможуть забезпечити стандартизованість різноманітних тестових артефактів впродовж всього життєвого циклу тестування, та підвищать зрозумілість для всіх членів команди. До прикладу, якщо розробка тестів відбувається на мові програмування Python, можна використовувати PEP8. Це керівництво по стилю для коду Python, яке рекомендоване до використання розробниками мови, хоч і не є обов'язковим. В багатьох інших мовах програмування є схожі набори правил і рекомендацій. Що стосується звітування, все дуже залежить за допомогою чого відбувається генерація репортів. На сьогодні дуже популярним є Allure Framework. Це гнучкий легкий багатомовний інструмент для створення тестових звітів, який не тільки показує дуже стисле представлення того, що було протестовано, у вигляді веб-звіту, але й дозволяє кожному учаснику процесу розробки витягувати максимум корисної інформації з повсякденного виконання тестів.

Але ці рішення не зможуть існувати повноцінно без процесу огляду та перевірки. Після розробки нових тест кейсів чи частин фреймворку, написаний код обов'язково повинен проходити етап огляду мінімум ще одним автоматизатором з команди. Це допоможе виявляти помилки та відразу їх виправляти, а також підтримувати наслідування прийнятих стандартів. Цей процес є описаний організацією ISTQB. Існує чотири типи рев'ю, від неформального, до найбільш стандартизованого і тому важливо розуміти коли і яке рев'ю застосовувати, адже цей процес відбирає час двох і більше інженерів одночасно.

1. Неформальне:

– може мати форму парного програмування або технічного керівника, який перевіряє дизайн та код;
– результати можуть бути задокументовані;
– корисність варіюється в залежності від рецензентів;
– основна мета: недорогий спосіб отримати певну користь;

2. Покрокове ознайомлення:

– зустріч під керівництвом автора;
– може відбуватися у формі сценаріїв, пробних запусків або групової участі;
– відкриті сесії;
– необов'язкова підготовка рецензентів перед зустріччю;
– необов'язковий звіт про рецензування;
– основні цілі: навчання, отримання розуміння, пошук недоліків;

3. Технічний огляд:

– задокументований, визначений процес виявлення дефектів;
– під керівництвом підготовленого модератора;
– підготовка рецензентів до зустрічі;
– необов'язкове використання контрольних списків;
– підготовка звіту про рецензування;

– на практиці може варіюватися від досить неформального до дуже формального;

– основні цілі: обговорення, прийняття рішень, оцінка альтернатив, виявлення недоліків, вирішення технічних проблем і перевірка відповідності специфікаціям, планам, правилам і стандартам.

4. Інспекція:

– під керівництвом підготовленого модератора;
– зазвичай проводиться як експертне оцінювання;
– збір метрик і різноманітних даних;
– формальний процес, що заснований на правилах та контрольних списках;
– визначені вхідні та вихідні критерії для прийняття програмного продукту;
– підготовка до зустрічі;
– звіт про перевірку, включаючи перелік висновків;
– формальний процес подальших дій;
– основна мета: виявлення дефектів.

Щоб досягти успіху при рев'ю, варто дотримуватись наступних правил:

– чітко визначати цілі;
– заповнювати та використовувати документи що прийняті на проєкті;
– залучати тільки тих людей які дійсно потрібні, щоб отримати максимальну користь і не витрачати час;
– намагатись виявити дефекти, але висловлювати їх об'єктивно;
– розглядати людські питання та психологічні аспекти;
– застосовувати відповідні техніки рецензування та рецензувати в атмосфері довіри;
– проводити тренінги з техніки проведення оцінювання;
– робити акцент на навчанні та вдосконаленні процесу;
– дотримуватись правил.

Важливо проводити регулярні перевірки коду тестових скриптів для виявлення невідповідности та відхилень від стандартів. Для цього потрібно використовувати статичні аналізатори коду для виявлення потенційних недоліків. Вони знаходять такі проблеми, як звернення до змінної з невизначеним значенням, неузгодженість інтерфейсів між модулями та компонентами, змінні, що не використовуються або неправильно оголошені, недосяжний (мертвий) код, відсутня та помилкова логіка (потенційно нескінченні цикли), надмірно складні конструкції, порушення стандартів програмування, уразливості в системі безпеки та порушення синтаксису коду та програмних моделей. Статичний аналіз потрібно обов'язково включати в загальний процес тестування, адже він покриває і знаходить ті проблеми, які не в змозі знайти динамічний підхід.

З точки зору компанії можливим рішенням є проведення тренінгів та семінарів, щоб автоматизатори ознайомлювались зі спільними стандартами та методиками, які використовуються в команді. Для вирішення проблеми відсутності стандартів та норм в автоматизації необхідно встановити спільні підходи та стандартизувати можливі процеси тестування, проводити регулярні перевірки коду та тренінги для автомати заторів [16].

Важливе питання, яке потрібно розглянути – інтеграція тестових фреймворків в процес Continuous Integration / Countinuos Development. Цей процес є важливим в розробці програмного забезпечення, адже він дозволяє автоматизувати процес запуску тестів та забезпечити швидке виявлення помилок. Однак, інтеграція з тестовими фреймворками має певні складнощі, які потребують уваги.

Відсутність зв’язку між тестовим фреймворком та системою управління версіями призводить до втрат часу при виявленні та виправленні помилок, оскільки немає можливості автоматично відстежувати зміни в коді та відповідні зміни в тестах. Розв’язання цієї проблеми є використання інструментів як Jenkins, які підтримують інтеграцію з системами управління версіями, до прикладу Git, що є найпопулярнішою системою. Це дозволяє автоматично запускати тести при зміні коду та відповідно виявляти та виправляти дефекти [17].

Відсутність можливості запуску тестів у різних середовищах. Програмне забезпечення часто призначається для роботи на різних операційних системах та з різними конфігураціями. Підходом до розв’язання є використання контейнерів, таких як Docker. Вони дозволяють створювати ізольовані середовища для запуску тестів. Це дозволяє забезпечити однакові умови для запуску тестів на різних середовищах та відповідно визначати та виправляти проблеми в різних конфігураціях. Контейнер – це стандартна одиниця програмного забезпечення, яка упаковує код і всі його залежності, щоб програма працювала швидко і надійно, незалежно від системи на якій він запущений. Образ контейнера Docker – це легкий, автономний, виконуваний пакет програмного забезпечення, який включає все необхідне для запуску програми: код, середовище виконання, системні інструменти, системні бібліотеки та налаштування. Переваги саме Docker контейнерів в тому, що вони стандартизовані (Docker створив галузевий стандарт для контейнерів, щоб їх можна було переносити куди завгодно), легкі (контейнери використовують ядро операційної системи машини, а тому не потребують окремої операційної системи для кожного додатка, що підвищує ефективність роботи сервера та зменшує витрати на сервер і ліцензування) та безпечні (програми в контейнерах безпечніші, а Docker надає найсильніші можливості ізоляції за замовчуванням в галузі).

Тестування займає багато часу, тому відсутність можливості запуску тестів паралельно теж може бути проблемою, адже це знижує швидкість розробки та випуску програмного забезпечення. Вирішенням проблеми є використання інструментів для паралельного запуску тестів, таких як Selenium Grid або TestNG, або схожих. Вони дають змогу запускати більшу кількість тестів одночасно, та відповідно скорочувати витрати часу на тестування. Важливо пам’ятати що автоматичні тести протрібно розробляти так, щоб вони не залежали один від одного, бо в іншому випадку паралельний запуск не принесе потрібної користі [18].

Ще одна важливе питання в процесі CI/CD – це можливість зберігати результати тестів, оскільки важко відстежувати звіти та визначати проблеми. Щоб вирішити це, потрібно використовувати інструменти збору даних про тести, таких як JUnit або TestNG. Ці інструменти дозволяють збирати таку інформацію про тести, як час виконання, статус і результати, та зберігати її у вигляді репорту. Це дозволяє проводити детальний аналіз результатів запуску.

Дослідження компанії DZone показало, що використання інструментів CI/CD забезпечує зниження тривалості тестування на 30–50 %, збільшення частоти випуску нових версій програмного забезпечення та зменшення кількості помилок, виявлених під час етапу тестування [19]. Це доводить, що інтеграція тестових фреймворків в процес CI/CD дозволяє знизити тривалість розробки та випуску програмного забезпечення, збільшити якість та надійність продукту та знизити витрати на розробку та тестування.

Компанія Sauce Labs також провела дослідження, яке показало, що використання контейнерів дозволяє скоротити час тестування в середньому на 25 %,

збільшити кількість тестів, які можна запустити паралельно, забезпечити більшу стабільність тестування та зменшення впливу на систему [20; 21].

Інтеграція тестових фреймворків в процес CI/CD є важливим етапом розробки програмного забезпечення, який дозволяє автоматизувати процес тестування та забезпечити швидке та ефективне виявлення помилок. Для розв'язання проблем, які можуть виникати під час інтеграції тестових фреймворків, можна використовувати інструменти CI/CD, контейнери для запуску тестів, інструменти для паралельного запуску тестів та збору даних про їх результати. Дослідження показують, що використання таких інструментів дозволяє знизити час тестування, збільшити якість та надійність програмного забезпечення та зменшити витрати на його розробку та тестування [22; 23].

**Висновки.** При розробці та використанні фреймворків для автоматизованого тестування існує ряд проблем, які впливають на роботу команди, якість тестування, його вартість та на ще багато факторів. Автоматизація приносить дуже багато користі, але за умови виправлення вищеописаних проблем під час планування та імплементації вищеописані або, якщо це не є можливим, їх вплив на проєкт мінімізується. В даній статті було розібрано найбільш поширені проблеми та описано способи їх розв'язання на основі досліджень різноманітних компаній та загальноприйнятих стандартів розробки та тестування. Хоча потреби різних проєктів дуже відрізняються, було наведено технології та підходи розв'язання можливих проблем, що можуть допомогти або підказати в якому напрямку шукати рішення конкретного питання. Вищеописані методи допоможуть збільшити ефективність та обсяги автоматизації, покращити якість кінцевого продукту.

## СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ:

1. Hardik S. Software Testing Cost. 2022. URL: https://www.simform.com/blog/software-testing-cost/ (дата звернення: 02.04.2023).

2. Dudekula M., Katam Reddy K., Kai P., Benefits and Limitations of Automated Software Testing: Systematic Literature Review and Practitioner Survey, Автоматизація тестування програмного забезпечення (AST). 7-й міжнародний семінар з питань, 2012.

3. Contan A., Dehelean C., L. Miclea, "Test automation pyramid from theory to practice", 018 Міжнародна конференція IEEE з автоматизації, якості та тестування, робототехніки (AQTR). Клуж-Напока, Румунія, 2018. С. 1–5, doi: 10.1109/AQTR.2018.8402699.

4. Vogel-Heuser B., Diedrich C., Fay A., Jeschke S., Kowalewski S., Wollschlaeger M. and Göhner, P. (2014) Challenges for Software Engineering in Automation. *Журнал програмної інженерії та додатків.* № 7. С. 440–451. doi: 10.4236/jsea.2014.75041

5. Shukla P., Patel D. Challenges in Test Automation Framework Design and Development. *International Journal of Advanced Research in Computer Science and Software Engineering*. 2016. С. 67–71.

6. Khan S. R., Ali T., Khan S. Issues and Challenges of Test Automation: A Systematic Literature Review. *Journal of Intelligent & Fuzzy Systems.* 2018. С. 2097–2108.

7. Твердохліб А.О., Коротін Д.С. Ефективність функціонування комп'ютерних систем при використанні технології блокчейн і баз данних. *Таврійський науковий вісник. Серія: Технічні науки.* 2022. № 6.

8. Цвик О.С. Аналіз і особливості програмного забезпечення для контролю трафіку. *Вісник Хмельницького національного університету. Серія: Технічні науки.* 2023. № 1.

9. Новіченко Є.О. Актуальні засади створення алгоритмів обробки інформації для логістичних центрів. *Таврійський науковий вісник. Серія: Технічні науки.* 2023. № 1.

10. Зайцев Є.О. Smart засоби визначення аварійних станів у розподільних електричних мережах міст. *Таврійський науковий вісник. Серія: Технічні науки.* 2022. № 5.

11. Humayun M., Iqbal M. Z. The challenges and benefits of continuous integration in software engineering. *Information and Software Technology.* 2017. С. 153–167.

12. Bajaj S., Singh S. Automation testing challenges and solutions: A review. *International Journal of Computer Applications*. 2017. № 173(4). P. 23–28.

13. Олександр Ш., Занурення в патерни проєктування / за ред. М. Ельвіри, Refactoring. Guru, 2021.

14. Leotta M., Clerissi D., Ricca F., Spadaro C. "Improving Test Suites Maintainability with the Page Object Pattern: An Industrial Case Study". IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops. Люксембург, 2013. P. 108–113, doi: 10.1109/ICSTW.2013.19.

15. Gunjan K., Page Object Model, URL: https://www.toolsqa.com/selenium-webdriver/page-object-model/ (дата звернення: 03.04.2023).

16. John Kent M. Sc, Test Automation: From Record / Playback to Frameworks, 2019, URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=327a028cc53b0774671ee380e0e268e3ffae28b4 (дата звернення: 03.04.2023).

17. Sauce Labs, Parameterized Testing: A Practical Guide for Better Tests., URL: https://saucelabs.com/resources/articles/parameterized-testing-a-practical-guide-for-better-tests .(дата звернення: 01.04.2023)

18. Tillmann Nikolai, de Halleux Jonathan, Xie Tao. Parameterized unit testing. 32nd ACM/IEEE International Conference on Software Engineering – Volume 2 (ICSE '10). Association for Computing Machinery. Нью-Йорк, США. С. 483–484. doi: https://doi.org/10.1145/1810295.1810441

19. Z. Ali, S. S. Awan, S. A. Khan, M. H. Shah, A Systematic Review of Test Automation Tools and Frameworks for Web Applications, 2019.

20. ISTQB Glossary, URL: https://glossary.istqb.org/en_US/search?term= (дата звернення: 30.03.2023).

21. Sheekha J, What is Version Control System, 2021, URL: https://www.toolsqa.com/git/version-control-system/ (дата звернення: 07.04.2023).

22. Dzone, The State of Continuous Integration and Continuous Delivery: 2021 Report, 2021, URL: https://dzone.com/articles/ci-cd-tools-and-trends-survey-2019-2020-results (дата звернення: 30.03.2023).

23. SauceLabs, The Benefits of Containers in Agile Testing, 2021, URL: https://saucelabs.com/resources/white-papers/containerization-testing-landscape-report-2019 (дата звернення: 30.03.2023).

**REFERENCES:**

1. Hardik S. Software Testing Cost, 2022. URL: https://www.simform.com/blog/software-testing-cost/ (data zvernennia: 02.04.2023).

2. Dudekula M., Katam Reddy K., Kai P., Benefits and Limitations of Automated Software Testing: Systematic Literature Review and Practitioner Survey, Avtomatyzatsiia testuvannia prohramnoho zabezpechennia (AST), 7-y mizhnarodnyi seminar z pytan, 2012.

3. A. Contan, C. Dehelean and L. Miclea, "Test automation pyramid from theory to practice", 018 Mizhnarodna konferentsiia IEEE z avtomatyzatsii, yakosti ta testuvannia, robototekhniky (AQTR), Kluzh-Napoka, Rumuniia, 2018, s. 1–5, doi: 10.1109/AQTR.2018.8402699.

4. Vogel-Heuser, B., Diedrich, C., Fay, A., Jeschke, S., Kowalewski, S., Wollschlaeger, M. and Göhner, P. (2014) Challenges for Software Engineering in Automation. Zhurnal prohramnoi inzhenerii ta dodatkiv, 7, s. 440–451. doi: 10.4236/jsea.2014.75041

5. Shukla, P., Patel, D., Challenges in Test Automation Framework Design and Development. International Journal of Advanced Research in Computer Science and Software Engineering, 2016, c. 67-71.

6. Khan, S. R., Ali, T., Khan, S., Issues and Challenges of Test Automation: A Systematic Literature Review. *Journal of Intelligent & Fuzzy Systems*, 2018, p. 2097–2108.

7. Tverdokhlib A.O., Korotin D.S. Efektyvnist funktsionuvannia kompiuternykh system pry vykorystanni tekhnolohii blokchein i baz dannykh. *Tavriiskyi naukovyi visnyk. Seriia: Tekhnichni nauky*, 2022, (6) [in Ukrainian].

8. Tsvyk O.S. Analiz i osoblyvosti prohramnoho zabezpechennia dlia kontroliu trafiku. *Visnyk Khmelnytskoho natsionalnoho universytetu. Ceriia: Tekhnichni nauky*, 2023, (1) [in Ukrainian].

9. Novichenko Ye.O. Aktualni zasady stvorennia alhorytmiv obrobky informatsii dlia lohistychnykh tsentriv. *Tavriiskyi naukovyi visnyk. Seriia: Tekhnichni nauky*, 2023, (1) [in Ukrainian].

10. Zaitsev Ye.O. Smart zasoby vyznachennia avariinykh staniv u rozpodilnykh elektrychnykh merezhakh mist. *Tavriiskyi naukovyi visnyk. Seriia: Tekhnichni nauky*, 2022, (5) [in Ukrainian].

11. Humayun, M., Iqbal, M. Z., The challenges and benefits of continuous integration in software engineering. *Information and Software Technology*, 2017, p. 153–167.

12. Bajaj, S., & Singh, S. (2017). Automation testing challenges and solutions: A review. International Journal of Computer Applications, 173(4), p. 23–28.

13. Oleksandr Sh., Zanurennia v paterny proiektuvannia / za red. M. Elviry, Refactoring.Guru, 2021.

14. M. Leotta, D. Clerissi, F. Ricca and C. Spadaro, “Improving Test Suites Maintainability with the Page Object Pattern: An Industrial Case Study”, 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops, Liuksemburh, Liuksemburh, 2013, s. 108–113, doi: 10.1109/ICSTW.2013.19.

15. Gunjan K., Page Object Model, URL: https://www.toolsqa.com/selenium-webdriver/page-object-model/ (data zvernennia: 03.04.2023).

16. John Kent M. Sc, Test Automation: From Record/Playback to Frameworks, 2019, URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=327a028cc53b0774671ee380e0e268e3ffae28b4 (data zvernennia: 03.04.2023).

17. Sauce Labs, Parameterized Testing: A Practical Guide for Better Tests., URL: https://saucelabs.com/resources/articles/parameterized-testing-a-practical-guide-for-better-tests .(data zvernennia: 01.04.2023)

18. Nikolai Tillmann, Jonathan de Halleux, and Tao Xie, Parameterized unit testing: 32nd ACM/IEEE International Conference on Software Engineering – Volume 2 (ICSE 10). Association for Computing Machinery, Niu-York, SShA, s. 483–484. doi: https://doi.org/10.1145/1810295.1810441

19. Z. Ali, S. S. Awan, S. A. Khan, M. H. Shah, A Systematic Review of Test Automation Tools and Frameworks for Web Applications, 2019

20. ISTQB Glossary, URL: https://glossary.istqb.org/en_US/search?term=(data zvernennia: 30.03.2023).

21. Sheekha J, What is Version Control System, 2021, URL: https://www.toolsqa.com/git/version-control-system/ (data zvernennia: 07.04.2023).

22. Dzone, The State of Continuous Integration and Continuous Delivery: 2021 Report, 2021, URL: https://dzone.com/articles/ci-cd-tools-and-trends-survey-2019-2020-results (data zvernennia: 30.03.2023).

23. SauceLabs, The Benefits of Containers in Agile Testing, 2021, URL: https://saucelabs.com/resources/white-papers/containerization-testing-landscape-report-2019 (data zvernennia: 30.03.2023).