DOI 10.31891/2307-5732-2022-309-3-70-74
УДК 004.4

БАГРІЙ Р. О.
Хмельницький національний університет
https://orcid.org/0000-0001-5219-1185
e-mail: bahriiro@khmnu.edu.ua
ПЕТРОВСЬКИЙ С. С.
Хмельницький національний університет
https://orcid.org/0000-0002-0590-0484
e-mail: petrovskijss@khmnu.edu.ua

# ОСОБЛИВОСТІ СУЧАСНОГО ТЕСТУВАННЯ ВЕБ-ДОДАТКІВ

*У роботі проведено дослідження технології та підходів до тестування веб-додатків в сучасних фреймворках. Розглянуто основні методи ручного тестування, що потребують великого обсягу ручної праці. Особливу увагу приділено технологіям автоматизованого тестування, що пропонується застосувати при розробці сайтів в сучасних фреймворках. Детально описана піраміда тестів та рівні тестування, проаналізовані основні переваги та недоліки для кожного рівня. Запропоновано застосовувати технологію розробки на основі поведінки (BDD) для автоматизованого тестування веб-додатків. Для підвищення ефективності процесу розробки веб-додатків застосовано шаблон проектування впровадження залежностей (DI) засобами, що вбудовані у сучасних фреймворках.*

*Ключові слова: комп'ютерні науки, тестування, фреймворк, веб-додатки*

Ruslan BAHRII, Serhii PETROVSKYI
Khmelnytskyi National University

## FEATURES OF MODERN WEB APPLICATION TESTING

*This paper investigates the technology and approaches to testing web applications in modern frameworks. The main techniques of manual testing, which require a large amount of manual work, are considered. The disadvantages of implementing the black-box testing method include the development of a formal specification, and the application of the white-box testing method requires analysis of software code.*

*Particular attention is given to automated testing technologies, which are proposed to be used to develop sites in modern frameworks. The Test pyramid and test levels are described in detail, and the main advantages and disadvantages of each level are analyzed. At the first level, a set of tests consists of unit tests that verify that an individual unit is working correctly according to the requirements of the specification. At the second level of the automation pyramid - integration tests check the interaction of a fragment of code with external components. At the third level - end-to-end tests perform testing of various user scenarios, and UI tests check the correct operation of the web application interface.*

*It is proposed to use behavioral development technology (BDD) for automated testing of web applications. BDD focuses on studying problems formulated based on customer stories and the construction of logic and tests based on these problems. BDD tests are best worked for integration testing, which involves testing different user scenarios.*

*A dependency injection (DI) design template has been used to improve the efficiency of the web application development process, with tools built into modern frameworks. This template applies the transfer of dependencies to an external, specially designed software component. Dependence is injected using a special IoC container.*

*Keywords: computer science, testing, framework, web applications*

### Постановка проблеми у загальному вигляді та її зв'язок із важливими науковими чи практичними завданнями

На даний час веб-додатки найбільш часто використовуються для надання послуг та продажу товарів в мережі Інтернет. Помилки, допущені у додатку, можуть завдати не тільки серйозного удару по репутації компанії, але й коштувати їй чималі гроші.

В той же час, тестування веб-додатків є складним та трудомістким завданням. Основна складність тестування полягає в тому, що розробка веб-додатків передбачає поєднання архітектурно складних і неоднорідних програмних фреймворків для клієнтською та серверної частини проекту. Особливості тестування веб-додатків також полягають у тому, що, зазвичай, сторінки динамічно змінюються та мають безліч станів в залежності від запитів користувача. Також нерівномірне навантаження на веб-додаток може викликати збої у роботі.

### Аналіз досліджень та публікацій

Тестування веб-додатку – це останній та обов'язковий етап технічної розробки програмного продукту. Він є важливим етапом процесі створення додатку, оскільки від якості тестування залежить подальший життєвий цикл додатку. Помилки на етапі тестування помилки призводять до додаткових витрат часу та ресурсів. Веб-додаток, який має помилки, викликає негатив у користувачів і, як наслідок, їх втрату. Як результат, замовник веб-додатку змушений платити за доопрацювання (а іноді за повторну розробку веб-додатку), а для розробників веб-додатку це втрата репутації.

Всі методи тестування веб-додатків, як і будь-якого іншого програмного продукту, поділяються на дві групи. До першої групи належать методи функціонального тестування або тестування методом «чорної скриньки» (black-box (BB)), при якому перевіряється лише зовнішня поведінка програми без аналізу

програмного коду. До другої групи належать методи структурного тестування, або тестування. методом «скляного ящика» (white-box (WB)), у якому перевіряються умови розгалуження, цикли та інші атрибути програмного коду. Прикладом тестування методом ВВ є система, в якій перевіряється відповідність реалізації веб-додатку специфікації. Генерація тестів у цій системі базується на формальному описі специфікації та моделюванні поведінки програми [1].

Застосування цих методів тестування пов'язане з великим обсягом ручної праці. Так, при реалізації методів тестування ВВ потрібно розробити формальну специфікацію, а за реалізації методів тестування WB необхідний аналіз програмного коду. Тому розробники тестів для веб-додатків прагнуть якнайбільше розширити сферу автоматичної генерації тестів.

**Формулювання цілей**

Дослідити та визначити проблеми автоматизованого тестування веб-додатків в сучасних фреймворках та запропонувати методи та підходи для їх вирішення.

**Виклад основного матеріалу**

**Автоматизація та піраміда тестів**

Автоматизація тестів дозволяє знайти помилки веб-додатку практично відразу після написання коду. Це дає можливість реалізувати швидку розробку веб-додатку при мінімальних затратах ресурсів на його тестування.

Для організації автоматичних тестів використовується концепція піраміди тестів Майка Кона: (Рис. 1):

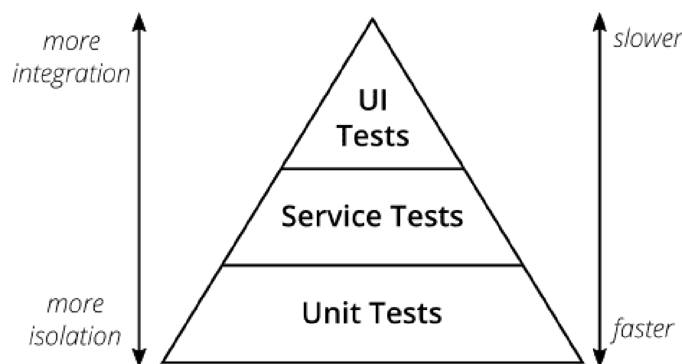


**Рис. 1. Піраміда тестів Майка Кона [2]**

Піраміда тестів складається з трьох рівнів: юніт-тести, сервісні тести (API тестування) та тести інтерфейсу користувача. При цьому швидкість виконання тестів та ізольованість об'єктів, що тестуються зменшується знизу вгору (від юніт - до тестів інтерфейсу користувача).

Для правильної організації тестування необхідно дотримуватись форми піраміди – розробити достатню кількість не великих за об‘ємом та швидких у виконанні юніт-тестів, декілька загальних тестів та обмежену кількість високорівневих наскрізних тестів, які перевіряють програму від початку до кінця.

Розглянемо ці рівні тестування більш детально.

**1. Юніт-тести.**

Основний набір тестів повинен складатися із юніт-тестів (модульних тестів). Вони перевіряють, чи окремий юніт (тестований об'єкт) працює належним чином відповідно до вимог специфікації. Кількість юніт-тестів у наборі повинна перевищувати кількість інших тестів.

Для тестування функцій виконується їх виклик з різними параметрами та з контролем очікуваних значень, що повертаються. Для об'єктно-орієнтованого підходу юнітом може бути метод чи цілий клас. Також під юніт-тестом можна розуміти фрагмент коду, який можна протестувати ізольовано від інших частин коду.

Сучасні мови програмування та фреймворки дозволяють ізолювати частину коду, що тестується, за допомогою контрольованої імітації цілих частин створюваного програмного продукту. Виклики класів замінюються на імітації (mocks) або заглушки (stubs). Реальний об‘єкт (наприклад, клас, модуль чи функція) імітується створеною копією. Копія має вигляд і діє як оригінал (дає такі ж відповіді на ті ж таки виклики методів), але це заздалегідь встановлені відповіді, які визначає розробник тестів для юніт-тесту. Для підтримки високої швидкості тестування юніт-тестів необхідно забезпечити відсутність контактів з БД, файловою системою та HTTP-запитами.

Однією з вимог для юніт-тестів є перевірка всіх нетривіальних розгалужень коду, включаючи сценарій за замовчуванням та пограничні ситуації.  При цьому виникає проблема у випадку рефакторингу коду –зміна внутрішньої структури коду без зміни зовнішньої поведінки. Таким чином втрачаєтеся важлива перевагу юніт-тестів: діяти як система безпеки для змін коду. Для того, щоб уникнути цієї проблеми необхідно не відображати у модульних тестуваннях внутрішню структуру коду, а проводити тестування поведінки, що спостерігається.

Структура модульних тестів складається з трьох дій - Arrange-Act-Assert [3]:

- Arrange – налаштування тестових даних;
- Act – виклик методу, що тестується;
- Assert – перевірка очікуваних результатів, що повертаються.

**2. Інтеграційні тести**

Модульні тести перевіряють невеликі фрагменти коду. Однак, щоб перевірити, як цей код взаємодіє з іншим кодом, необхідно запустити інтеграційні тести. По суті, це тести, які перевіряють взаємодію фрагмента коду із зовнішніми компонентами (базою даних, файловою система).

Інтеграційні тести є другим рівнем піраміди автоматизації тестування. Це означає, що його не слід виконувати так часто, як модульні тести. По суті, вони перевіряють, як функція взаємодіє із зовнішніми залежностями. Незалежно від того, чи це виклик бази даних чи веб-служби, додаток має ефективно спілкуватися та отримувати потрібну інформацію, щоб функціонувати належним чином.

Оскільки інтеграційні тести передбачають взаємодію із зовнішніми службами, вони виконуються повільніше, ніж модульні тести.

**3. Наскрізні тести / Тести UI**

На вершині піраміди знаходяться наскрізні тести. Вони гарантують, що весь додаток працює належним чином. Наскрізні тести виконуються найдовше, оскільки їм доводиться тестувати велику різноманітність сценаріїв користувача. Як і інтеграційні тести, ці тести також можуть вимагати від додатку зв’язку із зовнішніми залежностями.

Тестування програми від початку до кінця часто означає проходження через інтерфейс користувача.

Тести UI перевіряють правильність роботи інтерфейсу додатку. Дія користувача повинна ініціювати правильні події, дані повинні відобразитися користувачеві, стан UI має змінюватися очікуваним чином.

У веб-інтерфейсах бажано перевірити кілька аспектів UI: поведінку, верстку, юзабіліті, дотримання фірмового стилю та ін.

Тестування поведінки UI досить просте, так як потребує натиснення відповідник кнопок та уведення даних. Сучасні фреймворки для односторінкових додатків (react, vue.js, Angular та інші) мають інструменти та хелпери для ретельного тестування цих взаємодій на досить низькому рівні (в юніт-тесті). Для більш традиційного додатку з рендерингом на стороні сервера застосовуються тести на основі Selenium [3].

Цілісність верстки веб-програми перевірити складніше. Залежно від програми та потреб користувачів виникає необхідність перевірити, що зміни коду випадково не порушують верстку сайту.

Для перевірки зручності використання та дизайну застосовуються тестування юзабіліті.

Чим складніший UI, тим менш надійними стають тести. Помилки у браузері, проблеми із синхронізацією, анімація та несподівані спливаючі діалоги – характерні причини, які вимагають багато часу на налагодження тестів.

**Підходи до тестування**

До сучасних технології та підходів до тестування відносяться такі як: розробка через тестування (англ. Test Drive Development, TDD) та розробка на основі поведінки (англ. Behavior Driven Development, BDD).

Розробка через тестування (TDD) це технологія розробки програмного забезпечення, що починається з попереднього написання тестів, а потім пишеться код, що має проходити цей тест. Після цього виконується рефакторінг (покращення якості) коду до прийнятого стандарту (рис. 2) [4].

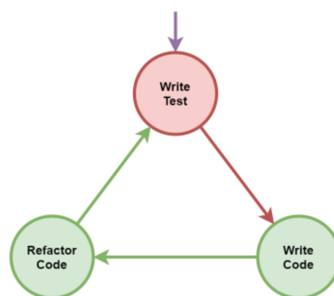


**Рис. 2. Цикл тестування TDD**

Розробка на основі поведінки (BDD) – це варіація розробки через тестування, яка дозволяє створювати якісніше програмне забезпечення та розробляти тест-кейси при тестуванні додатків відповідно до вимог замовника. В BDD основна увага приділяється дослідженні задач, що сформульовані на основі слів замовником та побудові логіки та тестів на основі цих задач [5].

Процес розробки проходить три етапи – *Discovery*: дослідження проблеми клієнта та можливість її вирішення, *Formulation:* формулювання прийнятих задач у вигляді бізнес-специфікації та *Automation:* автоматизація – створення відповідних автоматизованих тестів (рис. 3).

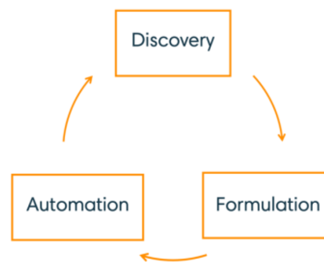


**Рис. 3. Етапи BDD-процесу**

Кожна з цих технологій має свої переваги та недоліки при автоматизованому тестуванні програмного забезпечення. Підхід TDD є сенс застосовувати для юніт-тестування, тобто, для перевірки роботи окремих функцій. Тоді як BDD – для інтеграційного тестування, тобто для перевірки, як окремі модулі працюють один з одним, що передбачає тестування різних сценаріїв користувача;

**Впровадження залежності**

Впровадження залежності для веб-розробки є один з найкращих патернів проектування, що дозволяє зробити програмний код більш читабельним та тестованим.

Впровадження залежності (англ. Dependency injection, DI) – шаблон проєктування програмного забезпечення, що передбачає передачу залежностей зовнішньому, спеціально призначеному для цього програмному компоненту [6].

Залежність у програмуванні краще показати на прикладі. Коли клас A використовує деяку функціональність класу B, тоді кажуть, що клас A залежить від класу B. Для використання методів інших класів, потрібно спочатку створити екземпляр цього класу, тобто клас A повинен створити екземпляр класу B. Таким чином, передача іншому програмному компоненту задачі створення об'єкта та подальше використання цієї залежності називається впровадженням залежностей.

Наприклад є клас ClassA, який містить екземпляр класу ClassB:

```
class ClassA {
  var classB: ClassB
}
class ClassB {
}
```

Без впровадження залежності ClassA може створювати залежні екземпляри класів при необхідності. Це зручно при програмуванні, але є багато недоліків. Так, ClassA та ClassB тісно пов'язані – замінити ClassB на інший немає можливості. Це не дозволяє провести модульне тестування ClassA, так як немає можливості ізолювати один клас від іншого.

Один з можливих шляхів вирішення проблеми є визначення всіх необхідних йому залежностей всередині конструктора та передачу відповідальності за створення залежних класів зовнішньому програмному компоненту:

```
class ClassA {
    var classB: ClassB
    constructor(classB: ClassB){
        this.classB = classB
    }
}
```

В такому випадку ClassA та ClassB пов'язані слабо і є можливість підмінити ClassB заглушкою та провести модульне тестування ClassA. До недоліків слід віднести підвищену складність коду, особливо при великій глибині залежних класів.

Сучасні фреймворки дозволяють застосувати механізм впровадження залежності та дозволяють визначити, як потрібно надати кожну залежність. Ін'єкція залежностей відбувається за допомогою IoC контейнера, який постачається разом із фреймворком.

**Висновки з даного дослідження і перспективи подальших розвідок у даному напрямі**

Проаналізовано наявні проблеми автоматизованого тестування веб-додатків в сучасних фреймворках. Виявлені проблеми, що виникають при тестуванні веб-додатків на різних рівнях тестування. Запропоновано застосовувати технологію розробки на основі поведінки (BDD) для автоматизованого тестування веб-додатків, що дозволяє проаналізувати вимоги замовника та побудувати логіку та тести відповідно до сформульованих вимог. Для підвищення ефективності процесу розробки веб-додатків запропоновано використати шаблон проектування впровадження залежностей (DI) засобами, що вбудовані у сучасних фреймворках.

**Література**

1. Van Veenendaal, Graham D., Rex B. Foundations of Software Testing: ISTQB Certification. 2019. 288 p.
2. Ham V. The Practical Test Pyramid. 2018. URL: https://martinfowler.com/articles/practical-test-pyramid.html.
3. Research Anthology on Agile Software, Software Development, and Testing. Information Resources Management Association. IGI Global, 2021. 2250 p.
4. BDD vs TDD vs ATDD: Key Differences. URL: https://www.browserstack.com/guide/tdd-vs-bdd-vs-atdd.
5. Behavior Drive Development (BDD) and Functional Testing. URL: https://medium.com/javascript-scene/behavior-driven-development-bdd-and-functional-testing-62084ad7f1f2.
6. Bojkić, Pržulj, Stefanović, Ristic. Usage of Dependency Injection within different frameworks. Conference: 19th International Symposium INFOTEH-JAHORINA At: Sarajevo. 2020. P. 119–124.