УДК 004.052
DOI: 10.18523/2617-3808.2020.3.62-68

Бенюх Л. І., Глибовець А. М., Афонін А. О.

ПОВЕДІНКОВИЙ ПІДХІД (BDD) ЯК ЕФЕКТИВНИЙ МЕТОД ДЛЯ ОРГАНІЗАЦІЇ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ У БЕЗПЕРЕВНОМУ ДОСТАВЛЕННІ ПРОДУКТУ

У роботі описано загальні відомості про організацію автоматизованого тестування та проаналізовано ефективність його застосування на проекті. Також досліджено різні підходи до організації автоматизованого тестування за допомогою таких методів, як написання коду через тестування, поведінковий підхід, підхід тестування за ключовими словами та набором даних. На підставі дослідження було побудовано систему автоматизованого тестування для покриття тестами графічного інтерфейсу користувача та вебсервісів (API тестування), використовуючи поведінковий підхід. Також було описано інструменти для організації поведінкового підходу тестування і переваги цього методу.

Ключові слова: поведінковий підхід (BDD), автоматизоване тестування, тестування графічного інтерфейсу, тестування вебсервісів (API тестування), система для побудови автоматизованого тестування.

Вступ

Цифрова революція диктує свої вимоги до бізнесу, який стосується багатьох сфер (онлайн-платформи з навчання, розваг, продажу, туризму тощо). Тому все більше підприємств вдаються за допомогою до різних інформаційних систем. У наші дні стрімко зросла потреба в автоматизації рутинних завдань, із метою мінімізувати та оптимізувати час і кошти. Тому стоїть глобальне завдання, як пришвидшити час на доставлення продукту до кінцевого користувача та при цьому не втратити якість продукту.

Саме автоматизація функціонального тестування відіграє дуже важливу роль у процесі отримання швидкого та ефективного результату, зокрема доставлення робочої версії продукту до кінцевого споживача у обмежений час.

Метою цієї роботи є дослідження та використання підходів до автоматизованого тестування у безперервному доставленні коду користувачеві у середовищі швидких змін і гнучкої методології розроблення.

У цій статті ми висвітлимо власний підхід до розв'язання поставленої проблеми за допомогою побудови системи, яка використовує поведінковий підхід для отримання ефективного тестування у безперервному доставленні коду.

Загальні положення автоматизованого тестування

Функціональне тестування допомагає забезпечити перевірку продукту згідно з вимогами шляхом тестування функціоналу програмного забезпечення. Саме автоматизація функціонального тестування відіграє дуже важливу роль у процесі отримання швидкого та ефективного результату тестування, зокрема доставлення робочої версії продукту до кінцевого споживача у обмежений час [17]. Основні цілі автоматизації тестування для організації безперервного доставлення продукту до користувача такі [17]:

- покриття тестами найбільш критичних із погляду бізнесу функцій, сценаріїв, бізнес-потоків і бізнес-процесів;
- покриття функціоналу, що розробляється у версії продукту, яку буде доставлено кінцеву користувачеві;
- покриття тестами вебсервісів (API);
- перевірка найважливіших із погляду бізнесу сценаріїв, бізнес-потоків та бізнес-процесів після змін у коді.

Процес побудови автоматизованого тестування

Покриття тестуванням пов'язане з багатьма діями у рамках програмного забезпечення і життєвого циклу розроблення продукту. На рис. 1 наведено рівні тестування.

Рис. 1. Рівні тестування

Залежно від основної мети, якої має досягти автоматизація тестування, виокремлюють різні ділянки функціоналу продукту для покриття автоматичними тестами на різних рівнях.

Визначальну роль під час побудови успішного процесу автоматизованого тестування відіграють такі етапи [2]:

1) побудова архітектури для автоматизації тестування;
2) аналіз можливості покриття автоматизованими тестами;
3) створення стратегії із побудови автоматизації тестування на проекті;
4) розробка системи з автоматизованого тестування.

Підходи для побудови автоматизованого тестування

Для реалізації ефективного автоматизованого тестування велике значення має підхід до побудови тестів, що залежить від самого процесу розроблення, бо ці речі мають бути взаємопов'язані. Виокремлюють такі основні підходи [7]:

- розроблення на основі тестів (*англ.* Test Driven Development). Цей підхід передбачає спочатку організацію автоматизованого тестування за допомогою написання модульних, функціональних та інтеграційних тестів, і лише після цього написання робочого коду продукту;
- поведінковий підхід до тестування (*англ.* Behaviour Driven Development), що є різновидом (доповненням) розроблення на основі тестів, із тією лише різницею, що цей підхід орієнтований на поведінку, яка покривається тестами (за тестового підходу основний фокус іде безпосередньо на сам код). Суть поведінкового підходу полягає у розробленні тестів, які описують систему у термінах, зрозумілих нетехнічному спеціалісту;
- тестування на основі ключових слів (*англ.* Keyword Driven Testing). Цей підхід передбачає використання ключових слів, що описують набір дій, потрібних для виконання конкретного кроку тестового сценарію;
- тестування на основі даних (*англ.* Data Driven Testing). За цього підходу тестові дані зберігаються окремо від тестових сценаріїв, наприклад, у файлах або в базі даних. Такий розподіл значно спрощує самі тести та їх об'єм.

Поведінковий підхід (BDD) як один з інструментів для організації автоматизації тестування

Часто витрачають багато часу на комунікацію між клієнтом і командою із розроблення для того, щоб зрозуміти, що потрібно розробити, що вже зроблено і що хочуть бачити кінцеві користувачі [17]. Цей зв'язок часто є слабким місцем у процесі. Поведінковий підхід є сукупністю практик для процесу розроблення, спрямованих на зменшення деяких ризиків, пов'язаних із комунікацією та зворотним зв'язком між командою з розробки та клієнтом. Саме за допомогою цього підходу є можливість імітувати, як програма має вести себе з погляду кінцевого користувача. Також тестові сценарії можна писати у реальному часі, спираючись на поведінку системи. Якщо команди з розробки працюють у змінному середовищі, яке базується на ітеративному доставленні програмного забезпечення до кінцевого користувача, то поведінковий підхід дає змогу проекту бути більш гнучким [7]. Кожен тестовий сценарій буде реальним сценарієм для користувача, а не простим тестовим випадком, завдяки впровадженню матриці «Роль-Поведінка-Причина» та формули «Дано-Коли-Тоді», що базуються на синтаксисі Gherkin.

На ринку є декілька інструментів із відкритим доступом для реалізації цього підходу. Розглянемо їх порівняння у таблиці.

Таблиця

Порівняльна таблиця інструментів BDD

Назва	«Жива» документація	Зручність у написанні тестів і документації	Налаштування та робота	Звітування	Інтеграція з CI/CD
Cucumber [4]	Наявна	Зручно, зі стандартами BDD	Зручний у використанні й нескладний у налаштуванні	Так, свій формат. У реальному часі	Так
Jbehave [11]	Частково	Зручно, зі стандартами BDD	Складний у налаштуванні	Так, але потребує налаштувань	Так
Gauge [8]	Частково	Не дуже зручно (без більш стандартного формату)	Складний у налаштуванні	Так, але потребує налаштувань	Так

Ми використали Cucumber, оскільки саме цей інструмент допоміг реалізувати такі потреби:

- взаємодія з бізнес-представниками та ефективний спосіб при отриманні зворотного зв'язку;
- зручність у написанні тестів для автоматизації;
- «жива» документація;
- візуалізація результатів тестування;
- реалізування системи безперервного доставлення продукту;
- упевненість у тому, що це правильний функціонал, затверджений із бізнес-представниками на ранньому етапі розроблення.

Розроблення системи для автоматизованого тестування

Систему автоматизованого тестування побудовано на базі фреймворку, який має такі рівні автоматизованого тестування:

- тестування вебсервісів – API тестування;
- функціональні автоматизовані тести графічного інтерфейсу користувача.

Система для автоматизації тестування має структуру програмного продукту. Як продукт, фреймворк для автоматизації тестування визначає функції програми, такі як оброблення зовнішніх файлів, взаємодія з графічним інтерфейсом, надає шаблони для структури тестів.

Опис функціоналу та самі тести були написані на основі поведінкового підходу, із використанням Cucumber і Gherkin синтаксисів.

Систему було інтегровано з сервісом безперервного доставлення коду Jenkins, а результатом кожного тестування є автоматично-згенерований звіт на основі Cucumber. Основною метою безперервної інтеграції (CI) є забезпечення швидкого зворотного зв'язку про стан процесу збирання. Сервер CI є програмою, яка постійно відстежує сховище вихідного коду проекту щодо змін. Кожного разу, коли відбувається зміна контролю над версією, сервер CI починає збирання (автоматично чи у ручному режимі) для компіляції та тестування цієї версії програми. Це означає, що всі автоматизовані тести виконуються для кожної нової версії бази коду.

Розглянемо, як запускається тест у Cucumber. Спочатку Cucumber розпізнає специфіку функціоналу, який описаний мовою користувача. Потім перевіряє функціонал на наявність сценаріїв для тестування та запускає ці сценарії. Кожен сценарій складається з набору кроків, які Cucumber обробляє. Саме завдяки цьому Cucumber може зрозуміти файли, де зберігається інформація про функціонал, базуючись на синтаксичних правилах. Синтаксичні правила, як уже згадано, мають назву Gherkin.

```
Feature: Sign up
  Sign up should be quick and friendly.
  Scenario: Successful sign up
    New users should get a confirmation email and be greeted
    personally by the site once signed in.
    Given I have chosen to sign up
    When I sign up with valid details
    Then I should receive a confirmation email
    And I should see a personalized greeting message
  Scenario: Duplicate email
    Where someone tries to create an account for an email address
    that already exists.
    Given I have chosen to sign up
    But I enter an email address that has already registered
    Then I should be told that the email is already registered
    And I should be offered the option to recover my password
```

Рис. 2. Приклад сценарію за допомогою синтаксису Gherkin

Розглянемо приклади написання сценаріїв за допомогою синтаксису Gherkin (рис. 2).

Ці тестові сценарії вносять у файл .feature, який можна додати у реальний проект разом із кодом. Організовувати файли .feature потрібно таким способом, щоб їх було легко знаходити та переглядати. Коли ви починаєте отримувати велику кількість файлів функцій у своєму проекті, важливо організувати у окрему папку, як на рис. 3.

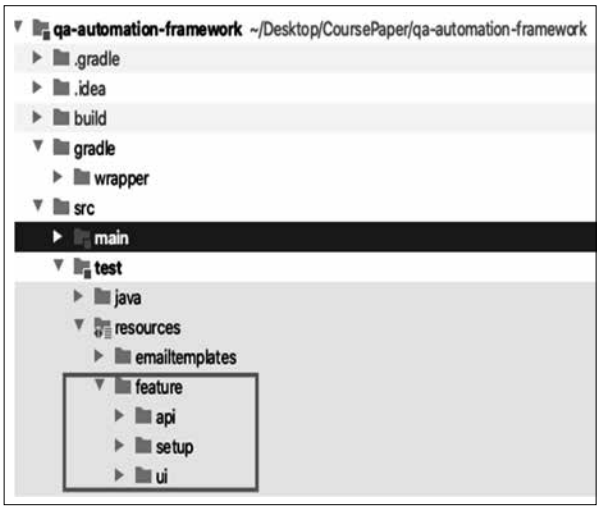


Рис. 3. Розташування .feature файлу

Отже, для того, щоб виконати ці дії, які наведені у тестових сценаріях, потрібно їх правильно оформлювати, а також додавати код (який виконуватиме дії та кроки) [17]. Кожен крок у файлах описується окремим методом з використанням Java. Опис кроків – це по суті код, який інтерпретують текст у .feature файлі. Код для розпізнавання кроків функціоналу міститься у папці test (див. рис. 4). Всі тести поділяють за напрямком тестування: API тестування або тестування графічного інтерфейсу (UI).

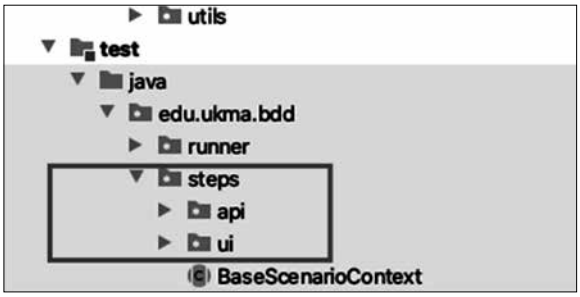


Рис. 4. Розташування опису тестових сценаріїв

У тестовому наборі перевірок визначення кроку можуть бути один чи два рядки коду, запуск якого йде за допомогою використання бібліотек для автоматизації тестування, як-от бібліотека для тестування графічного інтерфейсу користувача Selenium чи для вебсервісів (API тестування) RestAssured. Також потрібно зазначити, що фреймворк для автоматизованого тестування створено за допомогою PageObject підходу. Основна суть цього підходу полягає в тому, що окремо пишуть логіку для роботи з кожною сторінкою вебсайту. Спочатку ми імпортуємо бібліотеку Selenium для запуску та роботи з різними веббраузерами. Також клас наслідує від базового класу універсальні методи та містить додаткові методи для роботи на потрібній сторінці. Методи, своєю чергою, описують дії та локатори, за допомогою яких і здійснюється активність на сторінці пошуку та сам пошук. Тестові дані використовуються для створення тестових користувачів (у випадку, якщо їх не існує) та збереження їх до бази Redis у файлі testdata.properties. Для роботи безпосередньо з різними веббраузерами у папці utils написаний клас DriverUtility, що дає змогу створювати браузери з різними налаштуваннями.

Автоматичний запуск відбувається за допомогою сервісу Jenkins. Конфігурацію самого запуску тесту виконано за допомогою Jenkinsfile.

Подивімося, як відбувається взаємодія між компонентами системи та як організований потік (pipeline) запуску тестів і отримання результатів. Jenkins отримує тести з GitHub сховища. Потім запускає процес збірки проекту за допомогою gradle, який своєю чергою запускає Cucumber і Courgette. Останній дає змогу запустити тести у декілька потоків. Після закінчення тестування зберігається автоматичний звіт на базі інструментарію бібліотеки Cucumber. Весь описаний процес зображено на рис. 5.

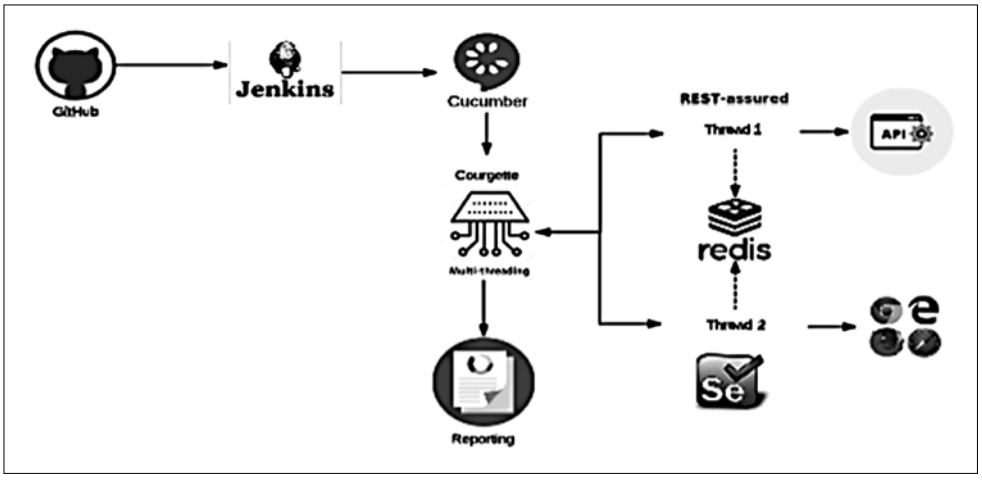


Рис. 5. Jenkins pipeline архітектура взаємодії компонентів

Візуалізація результатів тестування за допомогою Jenkins та Cucumber

Результати успішних і невдалих тестів можна отримати за допомогою бібліотеки Cucumber reports безпосередньо у системі Jenkins. Ця бібліотека генерує репорт після закінчення тестування на основі згаданих раніше тегів, наприклад @ui, @api.

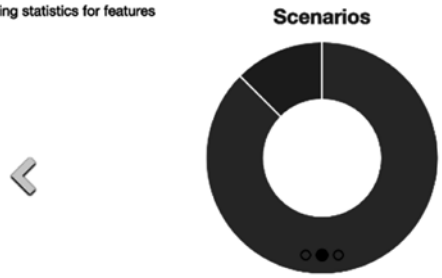
На рис. 6 наведено результат тестування за функціоналом, на рис. 7 – за окремим тегом; на рис. 8 – приклад невдало пройдених тестів.

Висновки

Дослідивши концептуальні підходи до організації автоматизованого тестування та поєднавши розроблення системи для її реалізації, можна з упевненістю стверджувати, що автоматизація тестування відіграє важливу роль в оптимізації безперервного доставлення інформаційного продукту до кінцевого користувача. Швидкий зворотний зв'язок допомагає виявити функціональні помилки на ранніх етапах

Features Statistics

The following graphs show passing and failing statistics for features

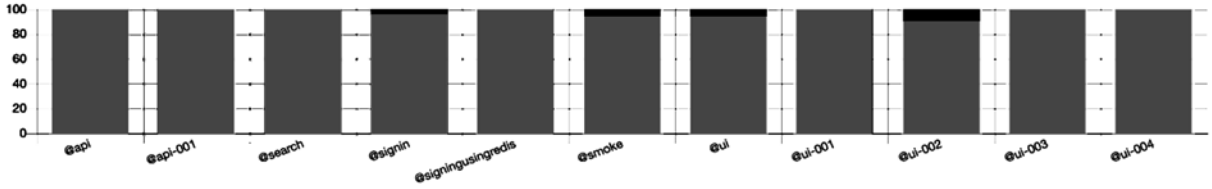


	Steps						Scenarios			Features	
Feature	Passed	Failed	Skipped	Pending	Undefined	Total	Passed	Failed	Total	Duration	Status
Feature to test searching scenarios	4	0	0	0	0	4	1	0	1	4.684	Passed
Feature to test sign in scenarios	25	1	0	0	0	26	3	1	4	36.592	Failed
Test Weather API.	12	0	0	0	0	12	3	0	3	5.903	Passed
	41	1	0	0	0	42	7	1	8	47.179	3
	97.62%	2.38%	0.00%	0.00%	0.00%		87.50%	12.50%			66.67%

Рис. 6. Результат тестування за функціоналом

Tags Statistics

The following graph shows passing and failing statistics for tags



	Steps						Scenarios			Features	
Tag	Passed	Failed	Skipped	Pending	Undefined	Total	Passed	Failed	Total	Duration	Status
@api	9	0	0	0	0	9	3	0	3	5.900	Passed
@api-001	9	0	0	0	0	9	3	0	3	5.900	Passed
@search	4	0	0	0	0	4	1	0	1	4.684	Passed
@signin	25	1	0	0	0	26	3	1	4	36.592	Failed
@signingusingredis	16	0	0	0	0	16	2	0	2	22.689	Passed
@smoke	38	1	0	0	0	39	7	1	8	47.176	Failed
@ui	29	1	0	0	0	30	4	1	5	41.276	Failed
@ui-001	4	0	0	0	0	4	1	0	1	4.684	Passed
@ui-002	9	1	0	0	0	10	1	1	2	13.903	Failed

Рис. 7. Результат тестування за тегами

Failures Overview

The following summary displays scenarios that failed.

Feature: Feature to test sign in scenarios
Tags: @ui @signin @ui-002 @smoke

Scenario Outline 6.643

Hooks

Steps

Given User is browsing 'Automation Practice' website	0.000
When User clicks **"Sign In"** on **"Homepage"**	2.376
And User lands on **"automationpractice.com/index.php?controller=authentication&back=my-account"** Page	0.005
And inputs email address **"test2@ukma.com"** and password **"incorrectpassword"**	4.256
Then User lands on **"automationpractice.com/index.php?controller=my-account"** Page	0.006

java.lang.AssertionError:

```
java.lang.AssertionError: User is not on expected page
        at org.junit.Assert.fail(Assert.java:89)
        at org.junit.Assert.assertTrue(Assert.java:42)
        at edu.ukma.bdd.steps.ui.SearchTest.user_lands_on(SearchTest.java:58)
        at ✱.User lands on "automationpractice.com/index.php?controller=my-account" Page(file:///Users/mykolak/.jenkins/workspace/Testing
```

Hooks

Рис. 8. Невдало пройдені тести

та підвищити загальний рівень якості системи, забезпечивши у такий спосіб високий рівень задоволення користувача.

До опису та реалізації тестового функціоналу можна підійти різними шляхами. Однак, як показало дослідження, оптимальним рішенням є використання поведінкового підходу (BDD), що дає змогу писати тести зрозумілою мовою для усіх сторін (бізнесу та інженерної команди), швидко, покращуючи розуміння та комунікацію між членами команди. Наприклад, тестові сценарії можуть написати навіть представники бізнесу у реальному часі, відповідно до своїх вимог.

Наступним важливим кроком є інтегрування системи автоматичного тестування з сервісом безперервного доставлення коду. Саме це інтегрування допомагає швидко виявити та виправити потенційні помилки функціоналу.

Поєднання поведінкового підходу до написання тестів з інтегруванням CI системи дає змогу досягти синергічного ефекту в оптимізації затрат часу та швидкості зворотного зв'язку під час тестування інформаційного продукту.

Список літератури

1. Amodeo Enrique. Learning Behaviour-driven Development with JavaScript / Enrique Amodeo. – Packt Publishing, 2015. – P. 7–24.
2. Bath Graham. Certified Tester. Advanced Level Syllabus Test Automation Engineer / Graham Bath, Rex Black. – Edinburg, TX: International Software Testing Qualifications Board, 2016.
3. Courgette JVM – 2020 [Electronic recourse]. – Mode of access: https://github.com/rgov/courgette-build.
4. Cucumber JVM – 2020 [Electronic recourse]. – Mode of access: https://github.com/cucumber/cucumber-jvm.
5. Docker – 2020 [Electronic recourse]. – Mode of access: https://www.docker.com.
6. Duval Paul M. Continuous Integration / Paul M. Duval. – Pearson Education, Inc, 2007. – P. 40–86.
7. Ferguson Smart John. BDD in Action / John Ferguson Smart. – Manning Publications Co, 2015. – P. 18–342.
8. Gauge – 2020 [Electronic recourse]. – Mode of access: https://gauge.org.
9. Gradle build tool – 2020 [Electronic recourse]. – Mode of access: https://gradle.org.
10. Grispin Lisa. A practical guide for testers and Agile teams / Lisa Grispin, Janet Gregory. – Pearson Education, Inc., 2009. – P. 100–380.
11. JBehave – 2020 [Electronic recourse]. – Mode of access: https://jbehave.org.
12. Jenkins. Build great things at any scale – 2020 [Electronic recourse]. – Mode of access: https://www.jenkins.io.
13. Rady Ben. Continuous testing with Ruby. Rails and JavaScript / Ben Rady, Rod Coffin. – North Carolina, US: Pragmatic Programmers, LLC., 2011. – P. 23–44.
14. Redis – 2020 [Electronic recourse]. – Mode of access: https://redis.io.
15. Rest Assured – 2020 [Electronic recourse]. – Mode of access: https://github.com/rest-assured/rest-assured/wiki/GettingStarted.
16. The Selenium Browser Automation – 2020 [Electronic recourse]. – Mode of access: https://www.selenium.dev/documentation/en/webdriver.
17. Wynne Matt. The Cucumber Book: Behaviour-Driven Development for Testers and Developers / Matt Wynne, Aslak Hellesoy. – Pragmatic Programmers, LLC, 2012. – P. 30–450.

References

Advanced Level Syllabus Test Automation Engineer. Edinburg, TX: International Software Testing Qualifications Board.

Amodeo, Enrique. (2015). *Learning Behaviour-driven Development with JavaScript*. Birmingham, UK: Packt Publishing

Bath, Graham, & Black, Rex. (2016). *Certified Tester*. Edinburg.

Courgette JVM. (n.d.). *Build script for Courgette, the binary patching tool used by Chrome*. Retrieved from https://github.com/rgov/courgette-build.

Cucumber JVM. (n.d.). *Cucumber for the JVM*. Retrieved from https://github.com/cucumber/cucumber-jvm.

Docker. (2020). *Docker Desktop*. Retrieved from https://www.docker.com.

Duval, Paul M. (2007). *Continuous Integration.* London, UK: Pearson Education, Inc.

Ferguson Smart, John. (2015). *BDD in Action*. Shelter Island, NY: Manning Publications Co.

Gauge. (2020). *Gauge Documentation*. Retrieved from https://gauge.org.

Gradle build tool. (2020). *Gradle Build Tool*. Retrieved from https://gradle.org.

Grispin, Lisa, & Gregory, Janet. (2009). *A practical guide for testers and Agile teams*. London, UK: Pearson Education, Inc.

JBehave. (2020). *References Guides*. Retrieved from https://jbehave.org.

Jenkins. (2020). *Build great things at any scale*. Retrieved from https://www.jenkins.io.

Rady, Ben, & Coffin, Rod. (2011). *Continuous testing with Ruby. Rails and JavaScript*. North Carolina, US: Pragmatic Programmers, LLC.

Redis. (2020). *Redis.*io. Retrieved from https://redis.io.

Rest Assured. (2020). *GettingStarted.* Retrieved from https://github.com/rest-assured/rest-assured/wiki/GettingStarted.

The Selenium Browser Automation Project. (2020). *WebDriver.* Retrieved from https://www.selenium.dev/documentation/en/webdriver.

Wynne, Matt, & Hellesoy, Aslak. (2012). *The Cucumber Book: Behaviour-Driven Development for Testers and Developers.* North Carolina, US: Pragmatic Programmers, LLC.

L. Beniukh, A. Hlybovets, A. Afonin

BEHAVIOUR DRIVEN TESTING AS AN EFFECTIVE APPROACH FOR AUTOMATED TESTING IN CONTINUOUS INTEGRATION

The global digitalization has transformed business dramatically, making it respond and adapt fast to the changing world and new challenges. The development teams face a global challenge of how to speed up product delivery time to the end user without losing product quality. Due to this, the demand for automation of routine tasks has significantly increased, because this process helps to minimize and optimize time and money for digital products delivery.

Functional automation testing plays a very important role in the process of faster and more effective product delivery to the end users in a limited time. For example, test automation plays a key role when you need to test periodically the same functionality over and over again to check whether the product is ready for delivery to the end user or not, especially, when working with very complex systems.

The development of the automated testing system architecture is influenced by such factors as the analysis of the possibility of coverage by automated tests, the creation of a strategy to build the automation testing on the project and the development of the automated testing system at its own. The test approach itself also plays an important role and could vary dramatically. For instance, Test-Driven Development, Behaviour-Driven Development, Keyword-Driven Testing, and Data-Driven Testing approaches.

The behavioural approach (BDD) has been used to develop an automation testing system described in this work. This approach is a set of practices aimed to reduce the risks with communication between the development team and the client. There is a wide range of tools for BDD-based testing, namely Cucumber, JBehave, Gauge, and more.

As a result, the automated testing system, described in this paper, has been built using Cucumber practices, which in turn allows functional verifications to be performed simultaneously at such levels as graphical interface (UI) and web service (testing API). Cucumber recognizes the specification for the tested functionality, which is described in a simple language, then looks whether testing scenarios are present in the feature files and only then runs test scripts. Each scenario consists of a set of steps that Cucumber processes and verifies against a given expected business logic.

It should also be noted that this system has been integrated into Continuous Integration, based on the Jenkins service. Test automation framework integration with CI tool helps to constantly monitor the project source code repository for changes, and this enables fast feedback and high quality of the product in the process of continuous development.

Keywords: behavioral driven development (BDD), automated testing, user interface testing, web services testing (API testing), continuous integration.

Матеріал надійшов 04.06.2020