DOI: https://doi.org/10.32839/2304-5809/2021-5-93-31
УДК 004.05

Каратанов О.В., Єна М.В.,
Бова Є.А., Устименко О.В.
Національний аерокосмічний університет імені М.Є. Жуковського
«Харківський авіаційний інститут»

ПОРІВНЯННЯ ПОПУЛЯРНИХ ТЕСТОВИХ ФРЕЙМВОРКІВ JUNIT И TESTNG

Анотація. Тестування вже давно є однією з найважливіших частин розробки програмного забезпечення. У зв'язку зі швидким зростанням популярності автоматизованого тестування, швидко почали розвиватися програмні продукти, які покликані допомогти з процесом автоматизації тестування, а саме фреймворки. У даній статті розглянуто найбільш популярних на даний час фреймворки для модульного тестування за допомогою мови програмування Java, проведено порівняльний аналіз двох найбільш популярних фреймворків та зроблено висновки за результатами їх порівняльного аналізу. Також, в даній статті розглянуто питання актуальності двох найбільш популярних фреймворків, а саме Junit та TestNG, ґрунтуючись на їх функціональних можливостях, піднімається питання актуальності їхнього функціоналу в сучасних реаліях, і при сучасних підходах до модульного та компонентного тестування й до життєвого циклу програмного забезпечення загалом. Всі порівняння, використані у статті, було зроблено ґрунтуючись на реальному досвіді використання фреймворків. Розглянуто головні функції фреймворків, використання яких базуються на Java-анотаціях і вбудованих параметрах фреймворків, виділено їх переваги та недоліки в JUnit і TestNG відповідно. У статті розглянуто не весь функціонал фреймворків, приведено лише базові функції, які реалізують більшу частину необхідного функціоналу для створення тестових випадків. Особливу увагу в даній статті приділено особливостям групування тестових випадків, заданню залежностей між тестами та розглянуто різні підходи до параметризації тестів (за допомогою створення *.xml файлу (файлів) в TestNG або за допомогою анотації в JUnit), розглянуто також можливості до ігнорування тестів, задання часу на ви-конання тест кейсів та задання передумов та постумов для кожного окремого теста за допомогою анотацій (@beforeSuite, @afterSuite – для створення перед-умов для цілих наборів тестів, @beforeTest, @afterTest – для створення умов для окремих тестів, @beforeMethod та @afterMethod – для створення умов до і після кожного тестового методу), а також задання передумов та постумов для тестових груп у цілому, розглянуто можливість задання порядку виконання тестів.

Ключові слова: тестування, Quality Control, Quality Assurance, модульне тестування, JUnit, TestNG, передумови, постумови, тестові дані, автоматизація тестування, Java, Java анотації, розробка, програмне забезпечення, контроль якості, якість програмного забезпечення.

Karatanov Oleksandr, Yena Maksym,
Bova Yegor, Ustumenko Oleksandr
Kharkiv National Aerospace University "Kharkiv Aviation Institute"

COMPARISON OF POPULAR TEST FRAMEWORKS JUNIT AND TESTNG

Summary. Testing has been one of the most important parts of software development for a long time. Due to the rapid growth in the popularity of automated testing, software products have been rapidly developed to help with the process of automated testing, namely frameworks. In this article were discussed most popular frameworks for modular testing using the Java programming language at this moment, also was done comparative analysis of two most popular frameworks and were drawn conclusions based on the results of their comparative analysis. Also, in this article was discussed a question of topicality of two most popular frameworks, namely Junit and TestNG, based on their functionality, raises the issue of their relevance in modern realities, and in modern approaches to modular and component testing and the software life cycle in general. All comparisons used in the article are based on real experience of using these frameworks. In article were discussed main functions of frameworks, which usage is based on Java-annotations and built-in parameters of frameworks, were considered, their advantages and disadvantages in JUnit and TestNG. In the article were not considered all the functionality of frameworks, only the basic functions that implement most of the necessary functionality to create test cases. Special attention in this article was paid to the peculiarities of grouping test cases, setting the dependencies between tests and considering different approaches to parametrization of tests (by creating * .xml file (files) in TestNG or by annotation in JUnit), also were considered the possibilities to ignore tests, setting time for test cases and setting pre-conditions and post-conditions for each test using annotations (@beforeSuite, @afterSuite – to create prerequisites for whole sets of tests, @beforeTest, @afterTest – to create conditions for individual tests, @beforeMethod and @ afterMethod – to create conditions before and after each test method), as well was considered setting pre-conditions and post-conditions for test groups of tests in general, the possibility of setting the order of tests.

Keywords: testing, Quality Control, Quality Assurance, module testing, JUnit, TestNG, precondition, postcondition, test data, automation testing, Java, Java annotations, development, software, software quality.

Постановка проблеми. Тестування вже давно є однією з найважливіших час-тин розробки програмного забезпечення. Вже давно відбувся перехід від ручного (мануального) тестування до автоматизованого, що дозволило збільшити швидкість і масштаби тестового покриття, збільшити зону контролю основних функцій програм на більш простому і швидкому рівні, а головне – автоматизувати рутинні мануальні завдання, які вимагають чіткого до-

тримання інструкцій і займають більшу частину роботи мануального тестувальника.

У зв'язку зі швидким зростанням популярності автоматизованого тестування, швидко почали розвиватися програмні продукти, які покликані допомогти з процесом автоматизації тестування.

Через це все складніше серед подібних фреймворків для тестування, підібрати багатофункціональний програмний продукт, який буде реалізовувати весь необхідний набір інструментів для вирішення завдань, які можуть з'явиться в сучасних реаліях.

Аналіз останніх досліджень і публікацій. Основними фреймворками для тестування є:

1) JBehave;
2) JUnit;
3) TestNG;
4) Selenide;
5) HttpUnit;
6) JWebUnit.

Але на даний момент існує лише два фреймворки, які активно використовуються при автоматизованому тестуванні. Порівняння фреймворків за популярністю можна побачити на рисунку 1.

Дослідженням даної теми займалися Нагаев Р. О. та Полевщик І. С. У своїй роботі «Автоматизация процесса тестирования программного обеспече-ния с приминением Junit» [7]. Подальший розгляд тестових фреймворків базується на «What is testing?» [2], Junit [3], ISO/IEC/IEEE 29119-1:2017 [4].

Виділення не вирішенних раніше частин загальної проблеми. Одним з найпопулярніших і найбільш використовуваних є JUnit версії 4 [1]. Даний фреймворк займає домінуючу позицію на ринку автоматизації, але з недавнього часу, вимоги до автоматизованого тестування почали стрімко зростати, і існуючі фреймворки (в тому числі і JUnit), перестали реалізовувати стрімко зростаючі вимоги програмістів і спеціалістів з тестування ПЗ. На зміну згаданому вище JUnit прийшов TestNG фреймворк [1].

JUnit – бібліотека для модульного тестування ПЗ на мові Java. JUnit належить сімейству фреймворків xUnit для різних мов програмування (МП) [3].

Бібліотека JUnit є одним з найстаріших фреймворків для тестування на Java (в даній статті розмова піде про порівняння фреймворків на МП Java), вона добре зарекомендувала себе в минулому.

TestNG – це тестове середовище для мови програмування Java, створене під впливом JUnit і NUnit. Метою розробки TestNG є можливість реалізації більш широкого діапазону категорій тестів: модульні, функціональні, інтегра-ційні і т. д. TestNG наділений більш потужними і простими у використанні функціями, більш гнучкими можливостями налаштування і великою кількістю вже реалізованих властивостей, у порівнянні з головним конкурентом JUnit [2].

Мета статті. В даній статті ставиться за мету виконати порівняльний аналіз двох найбільш популярних фреймворків [1], що використовуються на початок 2021 року, виділити їх переваги та недоліки.

Необхідно відзначити, що практично весь функціонал фреймворків побудований на Java анотаціях, тобто, являє собою мітку, спеціальну форму синтаксичних метаданих, які знаходяться в початковому коді й використовуються для його аналізу, компіляції або при його виконанні.

Для порівняння фреймворків можна виділити наступні категорії порівняння:

1) за гнучкістю маніпулювання з передумовою / постумовою;
2) за можливістю ігнорування тестів;
3) за можливістю «спільного виконання тестів»;
4) за можливістю параметризувати тести;
5) за можливістю обмежувати час виконання тестів;

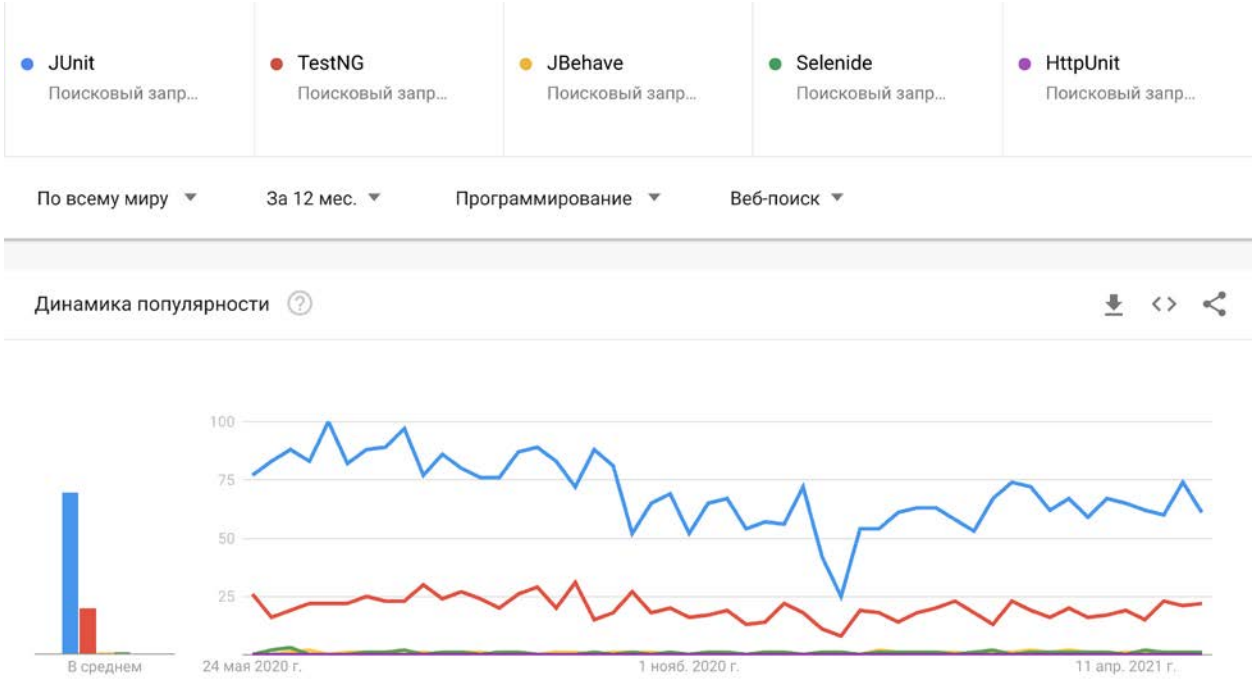

Рис. 1. Порівняння популярності фреймворків для тестування

6) за можливістю задання залежностей між тестами;

7) за порядком виконання тестових випадків [4].

Виклад основного матеріалу.

1 Порівняння фреймворків для тестування

1.1 За гнучкістю маніпулювання з передумовою / постумовою

Ініціалізацію і очищення тестів в JUnit можна реалізувати на двох рівнях, до і після кожного методу або ж після класу [4].

Передумова – це умови, які мають бути виконані клієнтом перед викликом операції [5].

Постумова – це умови, які мають бути виконані клієнтом після виклика операції [5].

У TestNG був реалізований такий же функціонал як в JUnit, але також було додано декілька анотацій, яких немає в JUnit, а саме: анотації на рівні тестового набору (@BeforeSuite, @ AfterSuite), а також конфігурації на рівні групи тестів (@BeforeGroup, @AfterGroup).

До основних відмінностями можна віднести наступне:

1) в TestNG механізм передумов, постумов реалізований більш гнучко, за допомогою вже згаданих вище анотацій @BeforeSuite, @ BeforeGroup. За допомогою цих анотацій, можна розбити існуючий тестовий набір (набір вхідних даних, умов виконання та очікуваних результатів [6]), на окремі групи або позначити їх окремою конфігураційною групою і вже стежити за даною групою окремо.

2) за допомогою більш широкого спектра анотацій, є можливість більш гнучко маніпулювати з тестами, наприклад, це може бути корисним при тестуванні одного й того ж функціоналу, використовуючи різні тестові набори. За допомогою анотації @BeforeSuite, можна задати або очищення даних з форми або наповнення форми даними, або ж використовувати різні конфігураційні налаштування для браузерів або для системи в цілому.

1.2 За можливістю ігнорування тестів

В даному пункті йдеться про таку можливість, як ігнорування тестів, а саме використання анотації @Ignore або ж властивості enabled.

Головна відмінність між двома фреймворками, полягає в тому, що в JUnit використовується анотація @Ignore, а в TestNG анотація @Test з параметром enabled, яка приймає значення або true або false, приклади використання анотацій, наведені на рисунках 2 і 3.

Загалом, даний функціонал в обидвох фреймворках ідентичний і дозволяє досягати однакового результату.

В даному випадку, обидва фреймворки рівні за можливістю ігнорування тестів.

1.3 За можливістю «спільного виконання тестів»

Даний функціонал присутній в обох фреймворках, але реалізована дана можливість зовсім по-різному.

В JUnit активно використовуються анотації @RunWith, @SelectPackages, @SelectClasses для групування тестів за деякими загальними характеристиками [4].

Під словом угруповання, слід розуміти об'єднання ряду тестів і виконання їх як єдиного великого тесту. Для об'єднання тестів і запуску

```
class testDifferenceBetweenTestNGAndJUnit {

@Ignore
@Test
public void checkCorrectSum() {
 int sum = 3 + 3;
 Assert.assertEquals(10,sum);
 }
}
```

Рис. 2. Використання анотації @Ignore (JUnit)

```
class testDifferenceBetweenTestNGAndJUnit {

    @Test(enabled = false)
    public void checkCorrectSum() {
        int sum = 3 + 3;
        Assert.assertEquals(10,sum);
    }
}
```

Рис. 3. Використання анотації @Test з параметром enabled

разом в наборі, необхідно використовувати анотацію @SelectPackages, для об'єднання класів @ SelectClasses.

В TestNG механізм спільного використання реалізовано за допомогою конфігураційного файлу з розширенням *.XML.

Цей конфігураційний файл містить у собі назву тестового набору (наприклад, набір для тестування позитивних сценаріїв або ж для smoke тестування).

У цей набір, додаються класи з тестами і потім вже, коли здійснюється виконання регресії або деякі зовнішні чи внутрішні перевірки, які необхідно здійснити для певної тестової групи (групи тестових заходів, які організову-ються та управляються разом, наприклад: тестування компонентів, інтеграційні тести, тест системи [6]), можна запустити на виконання XML файл, в якому вже містяться тести необхідної групи.

Це відразу дає ряд відчутних переваг, серед яких:

1) якщо тестові класи розкидані по всьому проекту, то щоб їх виконати їх необхідно спочатку знайти;

2) існує ризик, пропустити ряд класів, тим самим не перевірити / протестувати важливий функціонал;

3) важко запам'ятовувати, які тести вже були виконані, а які ні.

Приклад тестового набору продемонстрований на рисунку 4.

До недоліків JUnit можна віднести:

1) немає можливості створення зовнішніх файлів, в які можна помістити визначення цієї ж самої групи і надалі виконувати тільки цей файл, а не окремі класи, які будуть викликатися один за одним:

2) слід чітко розмежовувати типи анотацій, а саме, що і як слід об'єднати, не важливо що це: класи, методи або цілі тестові набори.

Часто існує необхідність виконання великої кількості тестів, як негативних, так і позитивних, тобто, тестів, спрямованих на перевірку різного функціоналу і різної логіки, часто одного і того ж компонента.

За допомогою створення груп, можна відмітити кожен тест (позитивний тест або негативний) і вже окремо спостерігати за їхньою поведінкою, відображати результат за допомогою зручних графіків або виводити статистику за допомогою AllureReport.

1.4 За можливістю параметризувати тести

Параметризація може бути корисною тоді, коли необхідно протестувати один і той же фрагмент з різними даними.

JUnit5 має деяку перевагу в тому, що методи тестування використовують аргументи даних безпосередньо з налаштованого джерела [4]. Розглянемо детальніше деякі анотації, що використовуються в JUnit:

1) @ValueSource, використовується як джерело даних і може працювати з масивами наступних типів даних: Short, Byte, int, long, float, double, char, string;

2) @EnumSource, використовується в якості джерела зумовлених раніше констант, які в подальшому будуть братися з перерахування і якось оброблятися тестом;

3) @MethodSource – як джерело даних буде використовуватися якась функція.

Важливим відмітити, що параметрами в JUnit можуть бути тільки примітивні типи, тобто, використання об'єктів у вигляді параметрів ніяк не передбачено в даному фреймворку.

В TestNG у якості джерела даних можна використовувати або @Parameters {параметр1, параметр2} або анотацію @DataProvider.

Розглянемо кожну анотацію більш докладно.

@Parameters може приймати параметри в дужках або ж через вже згаданий вище конфігураційний файл * .XML, приклад використання конфігураційного файлу для анотації, наведено на рисунку 5.

Якщо необхідно використовувати більш складні типи параметрів, то слід використовувати анотацію @DataProvider.

Для використання @DataProvider, необхідно:

1) створити статичний метод і позначити його анотацією @DataProvider;

2) задати ім'я для @DataProvider, наприклад, @DataProvider (name = "test");

3) помістити в анотацію @Test праметров (dataProvider = "test").

```
<suite name="TestSuite">

    <test name="Demo">

        <classes>

            <class name="com.fsecure.demo.testing.TestNGTest1" />
            <class name="com.fsecure.demo.testing.TestNGTest2" />

        </classes>

    </test>

</suite>
```

Рис. 4. Загальний вигляд тестового набору, який включає в себе два тестових класи

```
<suite name="My test suite">
    <test name="numbersXML">
        <parameter name="value" value="1"/>
        <parameter name="isEven" value="false"/>
        <classes>
            <class name="baeldung.com.ParametrizedTests"/>
        </classes>
    </test>
</suite>
```

Рис. 5. Використання параметрів у конфігураційному файлі

```
<suite name="My test suite">
    <test name="numbersXML">
        <parameter name="value" value="1"/>
        <parameter name="isEven" value="false"/>
        <classes>
            <class name="baeldung.com.ParametrizedTests"/>
        </classes>
    </test>
</suite>
```

Рис. 6. Використання @DataProvider

Використання @DataProvider продемонстровано на рисунку 6.

Це означає, що в якості даних, буде використовуватися саме те джерело даних, яке було заздалегідь створене і передане тесту. Важливим моментом є те, що таких джерел даних може бути багато, в такому випадку необхідно прибрати позначку static з джерела даних.

Цей інструмент є широко використовуваною функцією, тому що в джерелі даних, можна задати алгоритми генерації даних, обробки і передавати їх вже в тестовий метод.

Також можна в якості джерела даних використовувати дані з бази даних, які теж можна отримати в @DataProvider, а головною відмінністю від JUnit є можливість передачі об'єктів, як і передача примітивних типів.

Джерело даних (@DataProvider) може працювати і з об'єктами і з примітивами.

Цей механізм є революційним у модульному тестуванні та дозволяє створювати гнучкі алгоритми генерації даних і комбінувати різні підходи до наповнення тестів даними.

1.5 За можливістю обмежувати час виконання тестів

«Тайм-аути тестів» являються загальним для обох тестових фреймворків і дають можливість уникнути занадто довгого виконання тестів.

Це може бути корисним при виконанню складних SQL запитів, де вибірка багаточисленного масиву даних буде відбуватися довше необхідного часу, а в перспективі через це відбудеться збільшення часу виконання тестів.

У випадку з виконанням десяти, двадцяти тисяч тестів такі затримки можуть стати причиною збільшення часу виконання і розбору тестів на декілька годин.

Важливо відмітити, що для підтвердження дій використовуються JQM jobs. Виконання однієї такої роботи, може зупинити виконання всіх тестів і через подібні зупинки може вийти з ладу весь механізм тестового фреймворка. Тому, важливість даного механізму дуже суттєва, він допомагає запобігти небажаних затримок в неявних на перший погляд місцях, рисунок 7 демонструє, як різні JQM Job беруть дані з умовної бази даних кожні n секунд.

У результаті, якщо щось іде не так, робота завмирає і блокує доступ для інших робіт (job).

1.6 За можливістю задання залежностей між тестами

За допомогою механізму створення залежностей між тестами, можна створити ієрархію тестів, в якій кожний наступний тест буде заснований на по-передньому. Це не краще рішення з точки зору архітектури, так як, залежні тести складніше підтримувати.

У даних фреймворках головна відмінність між залежностями полягає в тому, що якщо початковий тест не пройде, то наступний тест, який залежить від умови проходження попереднтого або буде пропущено, як у випадку TestNG, або позначений як провальний, у випадку JUnit.

Великої різниці між даними підходами немає, тому що результат залишається одним, тести не пройдено, функціонал не перевірений і не покритий.

1.7 За порядком виконання тестових випадків

У JUnit механізму завдання порядку виконання тестів немає. Порядок виконання тестів випадковий і це в деяких ситуаціях може призвести до непередбачених результатів. В певних випадках, потрібно визначити який тест за яким

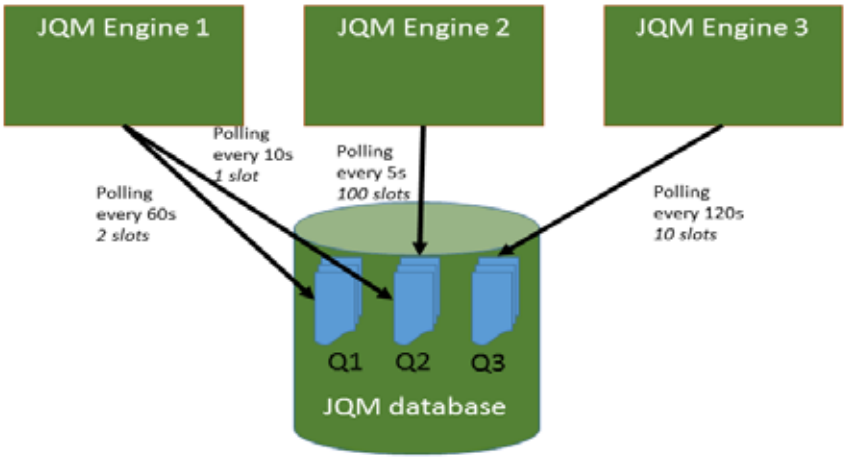

Рис. 7. Черга з JQM Job

буде виконуватися. Це може бути необхідним в разі перевірки реєстрації на деякому сайті. Наприклад, спочатку відбувається етап реєстрації, після нього процес підтвердження реєстрації і тільки потім вхід на сайт. У випадку, коли немає механізму пріоритезації тестів, можна вирішити проблему за допомогою з'єднання всіх трьох етапів в один і тим самим створити дуже великий, громіздкий, нечитаємий тест.

У TestNG дана можливість реалізована за допомогою параметру @Test (priority = № пріоритету виконання по порядку).

Даний механізм може бути необхідними в деяких випадках.

Висновки і пропозиції. Виходячи з описаного вище, можна зробити висновок, що TestNG є більш сучасним інструментом для модульного тестування.

JUnit позбавлений деякого суттєво важливого функціоналу для модульного тестування, а саме ряду механізмів, які стали на сьогодні одними з ключових і найбільш використовуваних, а саме, таких, як: параметризация тестів за допомогою об'єктів (dataProvider у TestNG має можливість працювати з об`єктами та примітивними типами, а jUnit у свою чергу може працювати лише з примітивними типами), задання пріоритету при виконанні тестів (TestNG має вбудовану анотацію @priority, яка вказує на порядок виконання тестів, у jUnit дана функція відсутня), створити тестових груп, завдання передумов та постумов для кожної з груп, анотації, які можна використовувати для створення передумов та постумов, а саме @beforSuite, @beforGroup, @ afterSuite, @afterGroup, спільне виконання тестів за допомогою xml файлу (ця функція реалізована у TestNG, та дає змогу виконувати пакети тестів запуском одного xml файлу). Такий файл може бути не один.

Базуючись на цих функціях, на даний час, TestNG надає та покриває більший тестовий функціонал для проведення модульного тестування, ніж jUnit. До того ж, TestNG охоплює весь функціонал ядра JUnit4.

TestNG призначений для функціонального тестування високого рівня і комплексного тестування інтеграції. Він особливо корисний для великих наборів тестів, на малих кількостях тестів різниця між фреймворками мінімальна, бо зі збільшенням кількості тестів, їх складності та різноманіття логіки легший підхід до розробки надає все ж таки TestNG.

Основні положення статті продемонстровані в таблиці 1.

Таблиця 1

Сравнение особенностей тестовых фреймворков TestNG и JUnit

	Pre/post Condition	Ignore test	Suit group test	Parametrize (primitive, object)	Time out Test	Dependency between test	Priority
TestNG	+	+	+	+	+	+	+
JUnit	+	+	+	+-	+	-	-

Список літератури:

1. Top 10 java testing frameworks. URL: https://www.lambdatest.com/blog/top-10-java-testing-frameworks/
2. What is testing? URL: https://testng.org/doc/
3. Junit. URL: https://junit.org/junit5/docs/current/user-guide/
4. ISO/IEC/IEEE 29119-1:2017. URL: https://docs.cntd.ru/document/1200134996
5. Лаврищева Е.М., Грищенко В.Н. Сборочное программирование. Основы индустрии программных продуктов : 2-изд., дополненное и переработанное. Киев : Наук. думка, 2009. 372 с.
6. Куликов С.С. Тестирование программного обеспечения. Базовый курс. Минск : Четыре четверти, 2017. 312 с.
7. Автоматизация процесса тестирования программного обеспечения с применением JUnit. URL: https://cyberleninka.ru/article/n/avtomatizatsiya-protsessa-testirovaniya-programmnogo-obespecheniya-s-primeneniem-junit/viewer

References:

1. Top 10 java testing frameworks. Access mode: https://www.lambdatest.com/blog/top-10-java-testing-frameworks/
2. What is testing? Available at: https://testng.org/doc/
3. Junit. Available at: https://junit.org/junit5/docs/current/user-guide/
4. ISO/IEC/IEEE 29119-1:2017. Available at: https://docs.cntd.ru/document/1200134996
5. Lavryshcheva E.M., Hryshchenko V.N. (2009) Assembly programming. Fundamentals of the software industry: 2nd ed., augmented and revised. Kyiv: Nauk. dumka, 372 p.
6. Kulikov S.S. (2017) Software testing. Basic course. Minsk: Chetire chetverti, 312 p.
7. Automation of the software testing process using JUnit. Available at: https://cyberleninka.ru/article/n/avtomatizatsiya-protsessa-testirovaniya-programmnogo-obespecheniya-s-primeneniem-junit/viewer