http://doi.org/10.35784/iapgos.8302 *received: 01.09.2025 | revised: 02.01.2026 | accepted: 28.01.2026 | available online: 30.06.2026*

COMPARATIVE ANALYSIS OF JAVA UNIT AND INTEGRATION TESTING TOOLS: JUnit, TestNG AND Spock

Dawid Grabek, Jan Gryta, Mariusz Dzieńkowski
Lublin University of Technology, Department of Computer Science, Lublin, Poland

Abstract. *The purpose of the study is to compare popular Java testing tools: JUnit, TestNg and Spock. The evaluation was con-ducted based using four criteria: performance, popularity, ease of learning, and code readability. A quantitative analysis was performed based on an experiment with predefined research scenarios, which included assertion tests, database operations, and computational tasks. Execution times of the scenarios were measured in various configurations of single-threaded and multi-threaded tests. The comparative analysis showed that no single tool is universally optimal. Spock demonstrates superior performance in terms of code readability and clarity. TestNG, by contrast, offers high execution performance and requires relatively little effort to learn and use effectively. JUnit stands out particularly in its support for multithreaded data access. The results therefore challenge the widely held belief in the JUnit framework's universality.*

Keywords: software testing, Java code testing, unit tests, integration tests, JUnit, TestNg, Spock

ANALIZA PORÓWNAWCZA NARZĘDZI DO TESTÓW JEDNOSTKOWYCH I INTEGRACYJNYCH JĘZYKA JAVA: JUnit, TestNG I Spock

Streszczenie. *Celem pracy jest porównanie popularnych narzędzi do wykonywania testów w języku Java: JUnit, TestNg i Spock. Weryfikacja została dokonana na bazie czterech kryteriów: wydajności, popularności, łatwości nauki oraz czytelności kodu. Analiza ilościowa została przeprowadzona na bazie eksperymentu i opracowanych scenariuszy badawczych, które obejmowały testy asercji, operacje bazodanowe i zadania obliczeniowe. Dokonano pomiarów czasów wykonania scenariuszy w różnych konfiguracjach testów jednowątkowych i wielowątkowych. Analiza porównawcza wykazała, że nie ma jednego, uniwersalnego, najlepszego narzędzia. Spock wypada najlepiej, gdy liczy się szybkość czytania i przejrzystość zapisu testów. Z kolei TestNg oferuje wysoką wydajność i wystarczy niewielki wysiłek, żeby szybko poznać to narzędzie i skutecznie nim się posługiwać. Natomiast JUnit wyróżnia się szczególnie obsługą wielowątkowego dostępu do danych. Wyniki podważają zatem powszechnie panujące przekonanie o uniwersalności frameworka do testów JUnit.*

Słowa kluczowe: testowanie oprogramowania, testowanie kodu Java, testy jednostkowe, testy integracyjne, JUnit, TestNg, Spock

Introduction

Software testing has always been an essential element of software development, providing assurance that the written code functions as the programmer expects. Given the current popularity of the Java language, numerous tools have emerged enabling simple and effective unit and integration testing within this ecosystem. Among the leading tools, JUnit, TestNG, and Spock stand out – each offering a different approach and set of functionalities.

This study focuses on conducting a comparative analysis of these three frameworks designed for testing Java code. The main objective of the research is to provide practical insights regarding their performance, popularity, ease of learning, and test code readability. This knowledge aims to assist programmers in making decisions when selecting optimal testing tools.

As part of the work, a research hypothesis was formulated, which assumes that JUnit, due to its dominant popularity, surpasses the other frameworks in all considered aspects.

1. Related works

In the software development process, unit and integration tests play a significant role throughout the entire application development process. Safe and stable applications require continuous testing at every stage of application development. The available literature for the studied tools contains a wealth of information and analyses containing valuable insights and conclusions. The literature cited in this chapter aims to better understand the testing problem in the scope of the studied frameworks, but also the testing process itself, in order to create a theoretical background for the implementation of the research conducted within this work.

As a result of the conducted literature review, only two studies were identified that directly compare the Java testing frameworks JUnit, TestNG, and Spock. Bielesza and Dzieńkowski [2] present an empirical analysis of these tools based on performance testing, functional evaluation, and an assessment of their popularity and community support. The authors indicate that JUnit 5 represents a stable and universal solution for the majority of projects, while TestNG is distinguished by its high flexibility in terms of configuration and control over test execution. Spock was assessed as the framework offering the highest readability and expressiveness of test code, at the cost of slightly lower performance.

Similar conclusions were drawn by Huber and Åkerblom [5], who conducted a comparative analysis of the same frameworks in the context of testing Spring Boot-based REST applications. Their results show that TestNG achieves the shortest test execution times, particularly in scenarios involving parallel test execution, whereas JUnit 5 provides a well-balanced compromise between performance, ease of configuration, and widespread adoption in engineering practice. Spock again stands out for its superior readability and conciseness of test code; however, it requires the use of the Groovy language, which may limit its applicability in production environments. In both studies, it is emphasized that differences in test execution time and resource consumption are not sufficiently significant to independently determine tool selection; instead, factors such as test readability, maintainability, and ecosystem maturity play a more critical role in practice. Consequently, the choice of a testing framework should be determined by project-specific characteristics, non-functional requirements, and the competencies of the development team.

Analysing other available publications, it can be observed that comparisons of Java testing frameworks often boil down to the two most popular tools: JUnit and TestNG. The study by Oshin and Chaudhary [11] presents a direct comparison of these two frameworks and evaluates them based on key features, including test protocols, test scenario parameterization options, configuration tagging, and report generation. The authors point to TestNG's advanced features in parameterization and built-in parallel testing support as main advantages. On the other hand, they also note JUnit's simplicity and wide usage as its strongest points, making it often the default choice for most development teams.

Numerous articles and internet sources contain characteristics and usage context for each tool. The JUnit framework is the oldest and most popular - testers use this tool not only because of its capabilities, but also because of the quality of the tests performed. Work by Ma'ayan [7] revealed typical problems

encountered when using JUnit in testing practice. These include inconsistent assertion error messages and high complexity of test methods when they contain too many assertions. TestNG, on the other hand, is presented as the most advanced tool and consequently, programmers and testers value it for the features it offers, which extend far beyond the basic scope of the JUnit environment. Works [11, 14] concern testing of multithreaded applications. They demonstrated that native support for test dependencies, grouping and advanced test lifecycle management along with built-in mechanisms for concurrent code testing make this tool stand out from others.

JUnit and TestNG are well described in specialized literature. Comparative analyses including the Spock framework, which is based on Groovy and encourages writing tests in a readable style similar to BDD (Behaviour-Driven Development), occur much less frequently. Although the developer community is showing increasing interest in the Spock framework, there are still few publications that systematically and empirically compare it with JUnit and TestNG in terms of aspects such as test performance under various conditions, ease of learning, and readability and maintainability of test code. It should also be emphasized that a significant part of the literature on test automation in the Java environment focuses on tools and frameworks created for testing web applications and introducing the BDD approach. In works [6, 12, 15, 16], studies and compilations of known tools such as Selenium, WebdriverIO, Katalon Studio, or QTP are implemented in relation to their effectiveness, ease of use, or included integration options. For example, the authors of studies [15] evaluated TestNG and WebdriverIO in the context of resource utilization and test execution speed in the GitHub application. Other scientists in work [4] studied the use of the Cucumber tool, supporting BDD, in Java and Kotlin languages, emphasizing its advantages in developing clear test specifications. Although these tools exceed the boundaries of unit and integration tests on which this work focuses, they show the extensive possibilities of available solutions, and the diversity of evaluation criteria used in practice.

A complement to analyses of specific tools are studies concerning broader issues related to the testing process, which provide examples and context of how and what factors to consider when conducting research. The authors of work [10] examine differences in effectiveness and characteristics of unit and integration tests based on the analysis of open-source projects, which emphasizes the importance of considering both levels of testing when evaluating frameworks. Issues related to quality and maintainability of test code, addressed in the context of test design patterns by researchers [3], directly connect with subjective aspects of evaluation, such as readability and ease of writing tests, which are also part of this study. Additionally, works [1, 13], analysing processes and criteria for selecting test automation tools in industry, indicate the continuous need to provide reliable, multi-aspect comparative data that could support making informed technological decisions.

Considering the aforementioned studies, a research gap can be identified, manifested by a very limited number of comparative analyses of the three frameworks: JUnit, TestNG, and Spock, as well as a limited number of publications devoted to testing using the Spock framework. Additionally, existing articles focus mainly on functionalities, and less on measurable performance and programmers' subjective feelings.

2. Research objective and hypothesis

The objective of the work is to perform a multi-aspect comparison of three popular tools for testing code in the Java language, namely JUnit, TestNG, and Spock. The study involves conducting unit and integration tests using these tools in various scenarios, checking performance, popularity, code readability, and learnability in the aspect of creating tests. Additionally, each tool was evaluated qualitatively by research participants.

For research purposes, a research hypothesis was formulated as follows: *In the environment of programmers using the Java language, it can be observed that JUnit has the greatest popularity, because this framework surpasses TestNG and Spock in many respects such as: test execution time, subjective feelings of testers, code readability, and ease of mastering the creation of correct tests.*

3. Methodology

To achieve the stated research objective and to verify the formulated hypothesis, an experimental research approach was adopted. The study focused on a comparative analysis of three Java testing frameworks: JUnit, TestNG, and Spock, conducted under identical and strictly controlled experimental conditions.

The experimental setup comprised a dedicated test application specifically designed and implemented to enable repeatable and reproducible experiments. For each framework, an equivalent set of test cases was executed, thereby minimizing the influence of confounding factors and ensuring the comparability of the obtained results.

In the experimental design, the independent variable was defined as the applied testing framework (JUnit, TestNG, or Spock), while the dependent variables encompassed metrics related to test performance and quality, including test execution time, system resource consumption, and selected characteristics of the test code. All remaining environmental and configuration parameters were maintained at constant levels throughout the experiments.

To conduct the experiment, a dedicated test environment was prepared and the measurement and monitoring tools were appropriately configured. The research procedure consisted of performing controlled experiments according to predefined test scenarios under three configurations: single-threaded execution and multi-threaded execution using four and eight threads. Each experiment was repeated multiple times in order to minimize the influence of random variability and to enhance the reliability and validity of the obtained results. In addition, the popularity, readability, and learnability of the selected testing frameworks were analysed.

The evaluation of the experimental outcomes was based on clearly defined evaluation criteria incorporating both quantitative and qualitative measures. The collected data were subjected to comparative analysis, which enabled the identification and interpretation of similarities and differences among the examined testing frameworks.

3.1. Research objects

The test application was designed using a modular architecture, which enabled the isolation of individual components and the execution of various research scenarios under different load conditions. This approach made it possible to systematically analyse the behaviour of the JUnit, TestNG, and Spock testing frameworks under diverse experimental conditions. The experiments were conducted between April and May 2025.

Within the developed test application, the following testing frameworks were analysed:

- JUnit – version 5.12.2 (latest stable release),
- TestNG – version 7.11.0 (current stable release),
- Spock – version 2.4-M6 (milestone release).

JUnit is currently the most widely used testing framework in the Java ecosystem. In many projects, it serves as the sole testing tool, which can be attributed to its maturity, ease of use, and strong integration with build tools and CI/CD systems [8].

TestNG is used less frequently than JUnit and often coexists with it within the same project rather than serving as a complete replacement. This usage pattern suggests that TestNG is primarily applied in specialized testing scenarios, such as parallel test execution or cases requiring more advanced configuration options [8].

In the case of the Spock framework, the requirement to use the Groovy language limits its adoption in production environments. Nevertheless, tests implemented using Spock are characterized by a smaller number of lines of code, and the use of the given-when-then structure significantly enhances readability and facilitates the understanding of test intent. Consequently, Spock is particularly advantageous in projects that adopt the BDD (Behavior-Driven Development) approach [5].

3.2. Tools

The following technologies were used to create the test application in Java:

- VisualVM 2.2 – a tool enabling monitoring of applications running in the Java virtual machine, used to conduct performance tests and analyse system resource usage,
- Docker Compose v2.35.1 – a platform that enabled executing tests in identical and isolated environments, which allowed obtaining reliable readings that will not be disturbed by any unwanted software,
- IntelliJ IDEA Ultimate 2025.1.4 – a popular development environment dedicated to this language, offering full support and integration with testing frameworks.
- Java Development Kit (JDK): Oracle JDK version 17.
- Apache Maven 3.9.12 – a build automation tool was employed to manage project dependencies and execute the test lifecycle.

3.3. Research environment and test scenarios

For the purpose of testing frameworks, an application was created containing a set of functionalities that allowed examining each tool separately under the same conditions. Each of the program's functions is associated with a corresponding research scenario. These scenarios addressed seven aspects, which are briefly discussed in Table 1.

Table 1. Research scenarios

No.	Description of tested feature
1	Testing assertion functions
2	Testing database operations
3	Testing computationally demanding operations
4	Testing performance of running a large number of tests
5	Comparison of the popularity of a given tool among programmers
6	Comparison of learnability of tool usage and test generation
7	Analysis and comparison of test readability

All tests were performed on a desktop computer with the configuration specified in Table 2.

Table 2. Test environment specification

Component's type	Description of tested feature
Processor	Intel(R) Core(TM) i5-11500H
RAM memory	16 GB 3200 MHz
Internal memory	SSD NVMe 512 GB
Operating system	Linux Ubuntu 24.04.02

3.4. Evaluation criteria

The essence of scenarios 1–4 was quantitative determination of the performance of the 3 frameworks, consisting of measuring the execution time of specified tests, which covered four aspects:

1) three different assertions: assertions of integers, strings, and exceptions,
2) database operations (adding, retrieving, and deleting),
3) computationally demanding operations (finding 500,000 prime numbers),
4) a large number of simple tests, consisting of single arithmetic operations or assertions.

The popularity analysis (scenario 5) was based on data placed on the mvnrepository.com website [9]. This portal contains a public search engine and catalogue of dependencies for JVM-based tools such as Maven or Gradle.

The evaluation criteria for scenarios 6–7 are based on subjective opinions and experience of the authors of this article in creating software in the compared technologies.

4. Results

In each of scenarios 1–4, three test sets were executed: single-threaded tests (described in Section 4.1) and multi-threaded tests using four and eight threads (described in Sections 4.2 and 4.3, respectively). Single-threaded tests were performed using the default framework configurations, whereas for the four- and eight-thread scenarios the configurations were modified to enable parallel execution with the corresponding number of threads. The collected data were subsequently averaged and normalized to the range 0-1 in order to facilitate comparison and to highlight differences between the evaluated frameworks.

4.1. Single-threaded tests

The comparison of single-threaded test execution for scenario 1 is presented in Figure 1. In this chart, a clear advantage of the Spock framework can be observed in scenarios testing assertion functions. The execution time of these tests was the shortest among the other two tools. In this case, the popular JUnit framework performed almost twice as poorly, achieving the lowest performance, i.e., the longest test execution time.

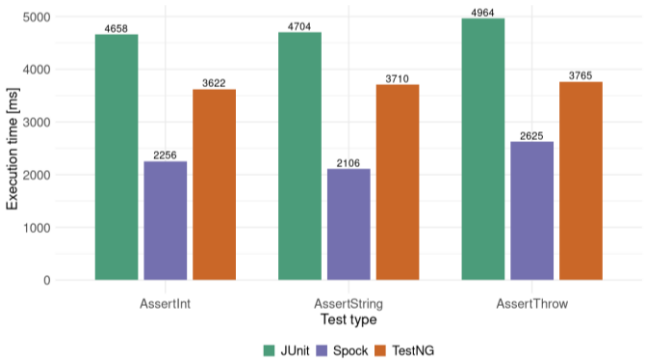


Fig. 1. Test execution time for scenario 1 in single-threaded configuration

For scenarios 3 and 4, Figure 2, i.e., for tasks requiring intensive calculations and tasks performed in parallel (Expensive Computation and Multithread task scenarios), all three frameworks achieved similar results, indicating comparable performance in these tests.

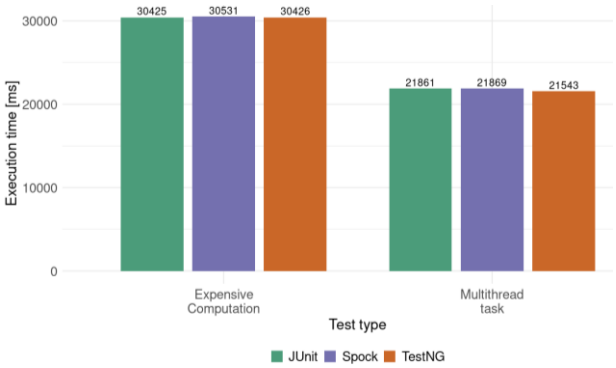


Fig. 2. Test execution time for scenarios 3 and 4 in single-threaded configuration

Small but noticeable differences also appeared in database operation tests, Figure 3. TestNG achieved the best results in them, showing the shortest execution time, while Spock achieved the highest values, indicating its lower performance in database interactions.

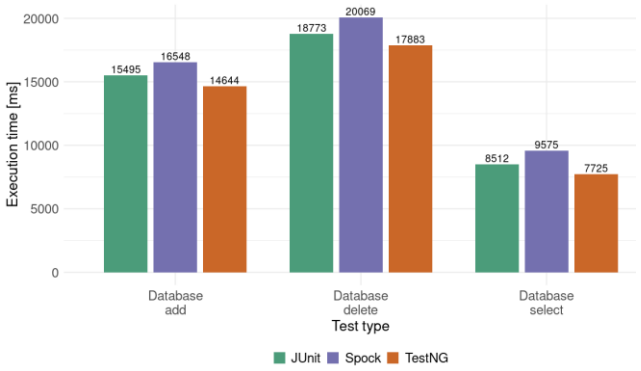


Fig. 3. Test execution time for scenario 2 in single-threaded configuration

Concluding this part of the research, the Spock framework clearly dominated in assertion tests in terms of speed, while achieving weak performance in database operations, which may be its significant limitation excluding it in such applications. TestNG presented the most balanced performance in different types of tests, and JUnit, despite poor results in assertion tests, handled both more complex and database operations quite well. Figure 4 presents the results of this part of the research in normalized form from 0 to 1, to enable easier comparison and highlight the differences occurring between frameworks in the results.

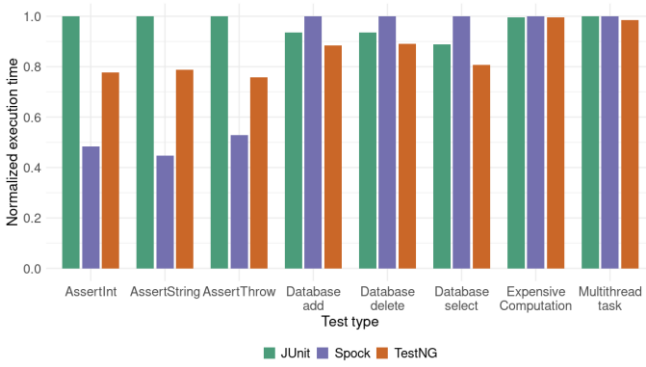


Fig. 4. Normalized test execution time for scenarios 1-4 in single-threaded configuration

4.2. Multi-threaded tests – 4 threads

Multi-threaded tests were also conducted in configurations for 4 and 8 threads. The Spock framework does not support multi-threaded code testing, so the presented charts show results only for JUnit and TestNG frameworks. Figure 5 shows the results of 4-thread performance tests for scenario 1.

Similarly to single-threaded tests concerning assertion functions, the JUnit framework performed worse in terms of average execution time. These differences, however, are not as pronounced as in previous single-threaded tests, but are still noticeable. This may mean that JUnit is more efficient when executing assertion tests in multi-threaded configuration.

In the case of complex operations and parallelized tasks, Figure 6, both tools achieved very similar results, and the differences between them were minimal.

For database tests, Figure 7, a slight advantage of JUnit over TestNG can be seen.

Summarizing this part of the research, both frameworks achieved similar results, however JUnit shows better performance in database tests, while TestNG has better performance in assertion tests. Figure 8 presents the results of this part of the research in normalized form from 0 to 1, to enable easier comparison and highlight the differences occurring between frameworks in the results.

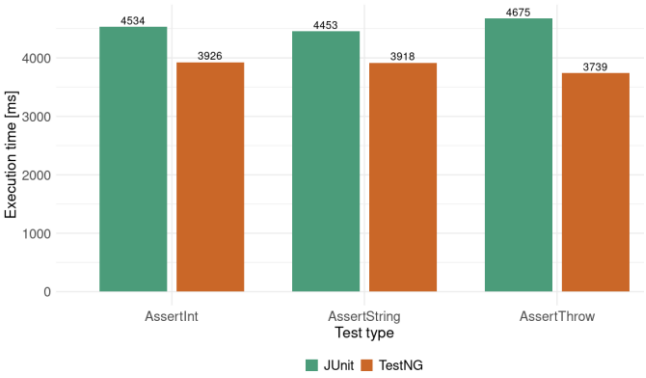


Fig. 5. Test execution time for scenario 1 in 4-thread configuration

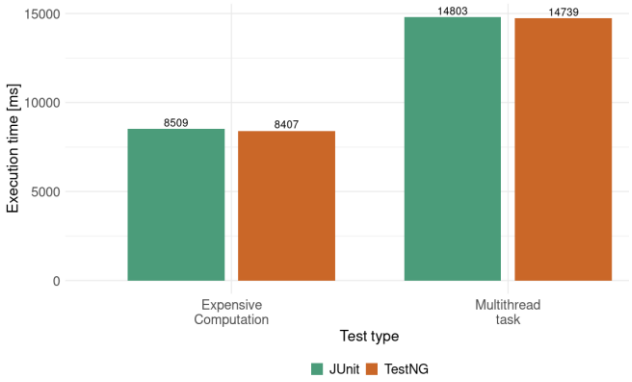


Fig. 6. Test execution time for scenarios 3 and 4 in 4-thread configuration

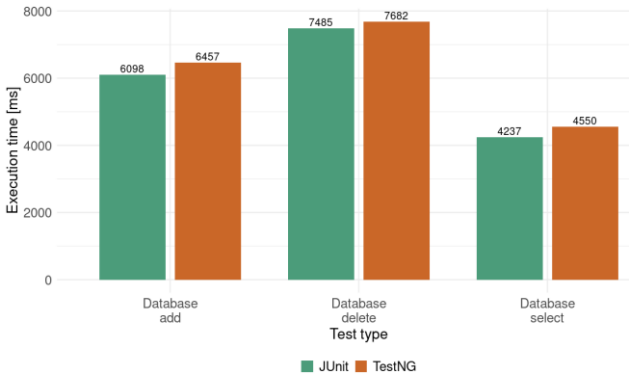


Fig. 7. Test execution time for scenario 2 in 4-thread configuration

Fig. 8. Normalized test execution time for scenarios 1-4 in 4-thread configuration

4.3. Multi-threaded tests – 8 threads

In this test, the Spock framework was also not considered, due to the fact that it does not support multi-threaded code testing. The chart in Figure 9 presents a comparison of results for JUnit and TestNG frameworks for tests from scenario 1 conducted in 8-thread configuration.

Similarly to single-threaded tests, in 8-thread configuration, JUnit is characterized by the highest execution time in all three assertion tests. TestNG again performed favourably in this scenario, achieving noticeably shorter execution times, confirming its higher performance in this category.

Fig. 9. Test execution time for scenario 1 in 8-thread configuration

In scenarios requiring large computational overhead (Expensive Computation) and in parallel operations (Multithread task), Figure 10, the results of both tools were almost identical, suggesting similar efficiency in handling intensive and parallelized processor tasks.

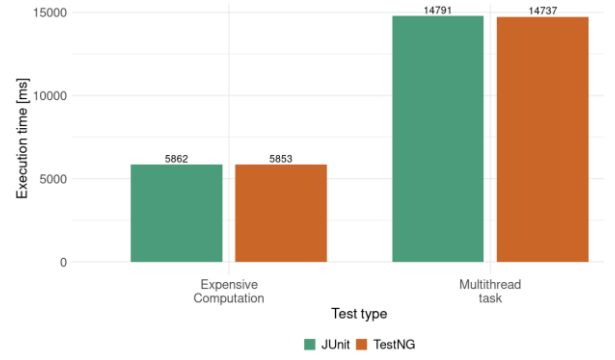


Fig. 10. Test execution time for scenarios 3 and 4 in 8-thread configuration

In the case of database interactions, a noticeable advantage in JUnit performance can be observed, Figure 11. This framework achieves better results in all three operations (add, select, delete), which may indicate better optimization of this tool for cooperation with databases in multi-threaded environments. TestNG, although it achieved worse results in database operations, its achievements are acceptable.

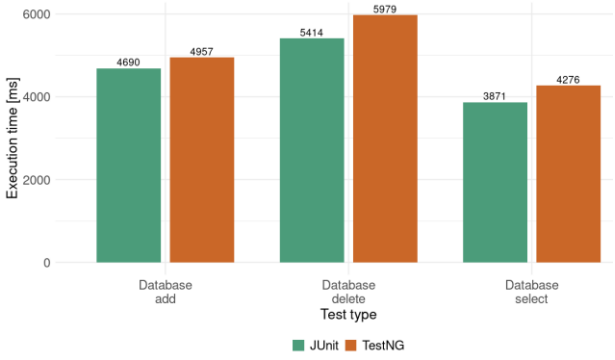


Fig. 11. Test execution time for scenario 2 in 8-thread configuration

Summarizing, the TestNG framework maintained its advantage in assertion tests also in a multi-threaded environment in configuration for 8 threads, while JUnit handled database operations better. Differences in computational and parallel scenarios were insignificant, which may indicate the maturity of both technologies in the context of handling computationally complex and multi-threaded tests. Figure 12 presents the results of this part of the research in normalized form from 0 to 1, to enable easier comparison and highlight the differences occurring between frameworks in the results.

Fig. 12. Normalized test execution time for scenarios 1-4 in 8-thread configuration

4.4. Popularity analysis

The chart in Figure 13 presents normalized data on declared usage of the evaluated tools for implementing tests based on Maven/Gradle used in IT projects by programmers.

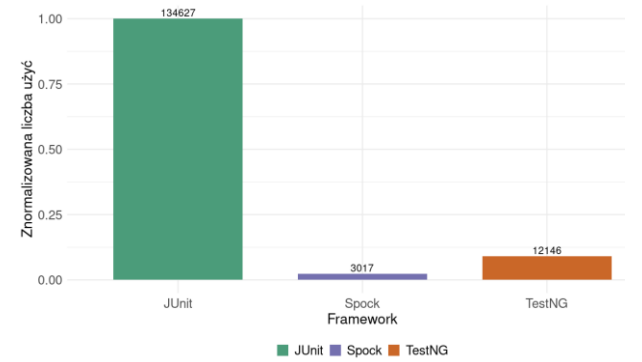


Fig. 13. Normalized results of Java testing framework popularity based on usage declarations

This chart shows a decisive advantage of JUnit in terms of declared usage of this tool by programmers and testers (over 134 thousand). On the other hand, Spock and TestNG were used many times less frequently, 3 thousand and over 12 thousand times respectively. These large differences prove that Junit as a testing tool is in the vast majority of cases the default choice by programmers.

4.5. Readability and learnability analysis

Each of the tested frameworks offers different programming experiences, both in terms of ergonomics and the so-called entry threshold, i.e., the difficulties that the technology poses for a beginning user.

Evaluating the learning curve, it was easiest to start working with TestNG. This framework draws good patterns from JUnit, adding its own simplifications and extensions, which makes learning to use this tool quick and smooth. The good documentation attached to this framework is its strong point and facilitates rapid tester implementation and efficient problem-solving.

JUnit, as the most popular tool in the field of Java code testing, also offers professional documentation and importantly benefits from enormous community support. Due to its dominant market position, it is easier to find solutions to encountered problems, and many examples and available libraries are by default adapted specifically to this framework.

Spock took the last place in the learnability category, which despite very good documentation, requires learning new syntax and understanding Groovy conventions, which may be a high barrier for many testing teams. However, in terms of test readability, Spock performed best. The Groovy language gives tests written in Spock very good readability – this framework enforces a clear test structure with distinct given, when, then sections, which significantly improves test code comprehensibility.

TestNG and JUnit performed very similarly in the code readability category. These frameworks allow maintaining appropriate test structure, but do not enforce it in any way. They transfer the responsibility for proper test structure to the programmer/tester.

Summarizing, test readability is definitely Spock's domain-its syntax and test structure are the most clear. TestNG and JUnit present a similar level in this regard, but TestNG gains a bit due to greater accessibility and faster learning and friendly documentation. Spock, although initially more demanding to learn, offers the highest test code clarity for those who decide to invest their time in it.

5. Conclusions

The aim of the conducted research was a multi-aspect comparative analysis of three popular frameworks for unit and integration testing in Java: JUnit, TestNG, and Spock. The goal was to provide a comprehensive picture of their capabilities, performance, and practical aspects of usage, to support programmers and testers in conscious selection of the tool best suited to project specifics and team requirements. The study covered both analysis of functionalities and syntactic features, as well as empirical performance tests and subjective evaluation of working with each tool.

In accordance with the adopted research methodology, a special test application was created and seven scenarios were developed, covering testing of assertion functions, database operations, tasks requiring large calculations, performance with a large number of tests, and analysis of popularity, learnability, and test code readability. Performance tests were conducted in controlled, isolated environments using Docker Compose, and the obtained results (execution time) were normalized, facilitating comparisons. Framework operation was tested in single-threaded and multi-threaded modes (for 4 and 8 threads, excluding Spock, which does not natively support multi-threaded tests).

The obtained quantitative analysis results showed diverse characteristics of the studied tools. Spock clearly surpassed the competition in assertion test execution speed in single-threaded mode. However, its efficiency in database operations proved to be lower. TestNG showed the best balance in performance in different types of single-threaded tests, which is partially consistent with observations by Oshin and Chaudhary [11], who noted that its advanced features may affect efficiency in complex scenarios. In multi-threaded tests, TestNG continuously maintained its advantage in assertion test speed, confirming its capabilities for concurrent code testing, also indicated in literature [14]. JUnit, despite relatively worse results in assertion tests, which may result from its simpler structure, performed well in more complex calculations and showed advantage in database operations in multi-threaded environments. In situations requiring intensive calculations and parallel tasks, the analysed frameworks (JUnit and TestNG in multi-threaded tests) achieved similar, statistically insignificant results. The popularity analysis, based on data from mvnrepository.com, clearly confirmed JUnit's leading role, which is consistent with the common perception of it as the default choice in the Java ecosystem.

The evaluation of subjective aspects of usage provided equally interesting observations. In terms of learnability, TestNG proved to be the simplest to learn, building on well-known JUnit patterns while simultaneously offering clear documentation and intuitive extensions. JUnit, due to its prevalence and enormous community support, also offers a relatively low entry threshold, which is often considered its advantage [11]. Spock, despite very good documentation, required learning Groovy syntax and unique conventions, which may pose some obstacle at the initial learning stage. Nevertheless, in terms of test code clarity, Spock proved to be unmatched. Its given-when-then structure and expressive Groovy language significantly increase test clarity. This observation fits into the growing interest in methods increasing test specification readability, such as Cucumber [4]. JUnit and TestNG provide a similar level of clarity, with greater responsibility for code organization resting on the programmer, which may result in difficulties in maintaining test quality, as emphasized by the author of publication [7] regarding JUnit.

The conducted analysis enabled verification of the stated research hypothesis, which suggested that JUnit, due to its popularity, would surpass TestNG and Spock in each analysed aspect. This hypothesis was partially refuted. While JUnit indeed leads in terms of popularity, providing solid support and a relatively simple start, in performance issues (especially in assertion tests) it yielded to TestNG and Spock. Also in code readability, Spock proved to be a tool that offers decidedly better solutions. TestNG, on the other hand, showed the best balance between advanced features, efficiency, and learning simplicity, making it a strong alternative to JUnit, as previous comparisons [11] had already suggested.

In conclusion, choosing the right testing tool in Java is a complex decision that depends on specific project requirements, team priorities, and programmer preferences, which has been confirmed in research on processes and criteria for tool selection in industry [1, 13]. JUnit remains a reliable, widely recognized standard, excellent for projects that prioritize simplicity and full compatibility. TestNG is an ideal solution for teams wanting more extensive features, greater control over tests, and appropriate performance, especially in complex situations. Spock represents an interesting alternative for projects where maximum readability and expressiveness of tests is crucial, even if it involves the need to acquire new technology. This work presents current empirical data and analyses that can support making this strategic decision, complementing existing literature, which most often focused on functional aspects, and less frequently on comprehensive performance evaluation and subjective feelings in direct comparison of these three specific frameworks.

References

[1] Abdulwareth, A. J., & Al-Shargabi, A. A. (2021). Toward a Multi-Criteria Framework for Selecting Software Testing Tools. *IEEE Access*, *9*, 158872–158891. https://doi.org/10.1109/ACCESS.2021.3128071

[2] Bielesza, G. W., & Dzieńkowski, M. (2023). Comparison of tools for creating and conducting automated tests. *Journal of Computer Sciences Institute*, *29*, 374–382. https://doi.org/10.35784/jcsi.3804

[3] Gonzalez, D., Santos, J. C. S., Popovich, A., Mirakhorli, M., & Nagappan, M. (2017). A Large-Scale Study on the Usage of Testing Patterns That Address Maintainability Attributes: Patterns for Ease of Modification, Diagnoses, and Comprehension. *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 391–401. https://doi.org/10.1109/MSR.2017.8

[4] Herman, I., & Plechawska-Wójcik, M. (2019). Study on applying the Cucumber tool in testing applications. *Journal of Computer Sciences Institute*, *11*, 91–95. https://doi.org/10.35784/jcsi.146

[5] Huber, L. (2023). *Comparing Spring REST api test frameworks—A comparison study* [Bachelor's thesis, Linnæus University]. https://lnu.diva-portal.org/smash/get/diva2:1764379/FULLTEXT01.pdf

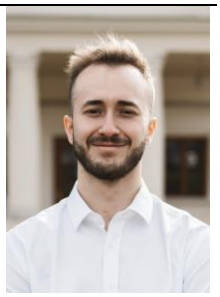
[6] Jagannatha, S., Niranjanamurthy, M., Manushree, S. P., & Chaitra, G. S. (n.d.). *Comparative Study on Automation Testing using Selenium Testing Framework and QTP*. Retrieved 31 August 2025, from https://www.academia.edu/8818002/Comparative_Study_on_Automation_Testing_using_Selenium_Testing_Framework_and_QTP

[7] Ma'ayan, D. D. (2018). The quality of junit tests: An empirical study report. *Proceedings of the 1st International Workshop on Software Qualities and Their Dependencies*, 33–36. https://doi.org/10.1145/3194095.3194102
[8] Madeja, M., Porubän, J., Chodarev, S., Sulír, M., & Gurbáľ, F. (2021). Empirical Study of Test Case and Test Framework Presence in Public Projects on GitHub. *Applied Sciences*, *11*(16), 7250. https://doi.org/10.3390/app11167250
[9] *MVN Repository*. (n.d.). [Data set]. Retrieved 31 August 2025, from https://mvnrepository.com
[10] Orellana, G., Laghari, G., Murgia, A., & Demeyer, S. (2017). On the Differences between Unit and Integration Testing in the TravisTorrent Dataset. *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 451–454. https://doi.org/10.1109/MSR.2017.25
[11] Oshin, V., & Chaudhary, V. (2018). Comparison Analysis of TESTNG and JUNIT frameworks for Automation with Java. *International Journal of Emerging Technologies and Innovative Research*, *5*(6), 48–54.
[12] Psujek, M., Radzik, A., & Kozieł, G. (2021). Comparative analysis of solutions used in Automated Testing of Internet Applications. *Journal of Computer Sciences Institute*, *18*, 7–14. https://doi.org/10.35784/jcsi.2373
[13] Raulamo-Jurvanen, P., Mäntylä, M., & Garousi, V. (2017). Choosing the Right Test Automation Tool: A Grey Literature Review of Practitioner Sources. *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, 21–30. https://doi.org/10.1145/3084226.3084252
[14] Shivaprasad, S., & Prasad, N. (2013). Unit Testing Concurrent Java Programs. *International Journal of Computer Applications*, *67*(10), 41–46. https://doi.org/10.5120/11435-6798
[15] Shtokal, A., & Smołka, J. (2021). Comparative analysis of frameworks used in automated testing on example of TestNG and WebdriverIO. *Journal of Computer Sciences Institute*, *19*, 100–106. https://doi.org/10.35784/jcsi.2595
[16] Wac, A. D., Watras, T. K., & Kozieł, G. (2020). Comparative analysis of solutions used in automated testing. *Journal of Computer Sciences Institute*, *15*, 156–163. https://doi.org/10.35784/jcsi.2048

M.Sc. Dawid Grabek
e-mail: dawid.grabek@pollub.edu.pl

Dawid Grabek received his engineering degree in computer science at Lublin University of Technology. He is particularly interested in web development and databases. He enjoys exploring technologies such as Java, TypeScript, React, Node.js, SQL, and containerization tools, with focus on scalable applications.

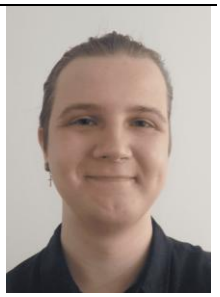
https://orcid.org/0009-0007-4988-0896



M.Sc. Jan Gryta
e-mail: jan.gryta@pollub.edu.pl

Jan Gryta received is engineering degree in computer science at the Faculty of Electrical Engineering and Computer Science at the Lublin University of Technology. The author's research interests include Java, Groovy, grails and web technologies.

https://orcid.org/0009-0002-4459-3867



Ph.D. Mariusz Dzieńkowski
e-mail: m.dzienkowski@pollub.pl

Assistant professor in the Computer Science Department at the Faculty of Electrical Engineering and Computer Science at Lublin University of Technology. His scientific interests include human-computer interaction, accessibility and usability, eye tracking applications and web application development.

https://orcid.org/0000-0002-1932-297X