JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 25 (2022) 309–314
Received: 16 June 2022
Accepted: 2 July 2022

# Evaluation of Flutter framework time efficiency in context of user interface tasks

# Ocena wydajności czasowej frameworku Flutter w kontekście obsługi interfejsów użytkownika

Damian Białkowski*, Jakub Smołka

*Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland*

**Abstract**
The article describes a comparative analysis of the time performance between native Android applications (created with the Android SDK and Java language) and applications created with the Flutter multi-platform framework. The study consisted of creating three pairs of applications that are functionally identical with each other using both programing tools, and then examining the time taken to perform individual actions by both applications. The functionality of the test applications consisted mainly of tasks related to operating on the user interface. The study was carried out on a Huawei P30 phone using the Perfetto tool. Results confirm that native apps are more time efficient than Flutter apps.

*Słowa kluczowe*: Flutter; szkielet wieloplatformowy; aplikacja mobilna; Android

**Streszczenie**
Artykuł opisuje analizę porównawczą wydajności czasowej aplikacji natywnych systemu Android (stworzonych za pomocą Android SDK oraz języka Java) oraz aplikacji stworzonych za pomocą wieloplatformowego frameworku Flutter. Badanie polegało na stworzeniu trzech par identycznych ze sobą funkcjonalnie aplikacji za pomocą obu rozwiązań, a następnie zbadaniu czasu wykonania poszczególnych działań przez obie aplikacje. Funkcjonalność aplikacji testowych składała się głównie z zadań z zakresu operowania na interfejsie użytkownika. Badanie zostało przeprowadzone na smartfonie Huawei P30 za pomocą narzędzia Perfetto. Wyniki potwierdzają lepszą wydajność czasową aplikacji natywnych względem aplikacji Fluttera.

*Keywords*: Flutter; cross – platform framework; mobile app; Android

*Corresponding author
*Email address*: damian.bialkowski@pollub.edu.pl (D. Białkowski)



## 1. Introduction

Cross-platform technologies are one of the most popular development directions of IT industry. The essence of these solutions is the ability to create applications for many different environments using only one code base. This solution has two obvious advantages. Firstly, it significantly reduces the time needed to deliver the end product to all platforms, which translates into the amount of resources needed to complete the project, and thus the cost of product delivery. The second advantage is that a multi-platform software developer does not need to know all native technologies for each target platforms.

Flutter framework is being developed by Google corporation and was released in 2017. This tool allows for creating Web, mobile (Android and iOS) as well as desktop and embedded applications. It uses Dart as its programming language. Dart is a language optimized for user interfaces creation.

Cross-platform frameworks have significant advantages over native solutions of specific environments, however there are concerns regarding their quality and usability. The objections mainly concern the efficiency of multi-platform solutions, as well as the possibility of obtaining the same functionalities and appearance of the application as in the case of native applications.

In case of mobile applications performance testing, researchers usually inspect such actions as: sorting data, database writing and reading or downloading file from the Internet. This article, however, describes the less frequently studied aspect of mobile applications, which is the display and manipulation of user interfaces. In regard to the Flutter framework, it is even more important, because the manufacturer strongly emphasizes its usefulness as a tool for building user interfaces.

The purpose of the study is to determine whether there is a difference between the time efficiency of an application created with the Flutter framework and the time efficiency of a native Android application (created with the Android SDK and Java language). Tasks performed by applications consists of user interface operations.

## 2. Literature review

Despite the short presence of the Flutter framework on the market, first scientific publications about this tool are gradually appearing.

The Authors of the article [1] conducted a multi-criteria analysis of applications written in the Flutter framework. Execution times of the following operations were examined: writing and reading data from a file, sorting data table, writing and reading data from a local

database. Results were compared to native Android application. In most cases, the native application has proven to be more efficient or as efficient as the application developed with Flutter. Additionally, the size of the source code for both applications was examined. However, no tests have been undertaken on the display of the user interface.

In the article [2], the Authors examined the performance differences between mobile applications of Android and iOS systems created with the following tools: Flutter, React Native and native languages, respectively Java for Android and Swift for iOS. The execution time of the following tasks was measured: HTTP query with the retrieval of response in JSON format, displaying the first 5 elements of the list on the screen, writing 7 elements of the list to the local database and reading 5 elements from this database. On both platforms native applications performed list display and database read tasks faster. Native applications of iOS also performed data retrieval via HTTP protocol faster. In the remaining cases, there was no significant difference in execution time or multi-platform applications completed faster. The authors indicated that cross-platform frameworks often match the performance of native applications.

An example of using the Flutter framework application in a real IT system is presented in the article [3] in which the Authors created a system to support shipment tracking by using the GPS technology. The system included both viewing shipments by customers and handling the status of the package by the supplier. It consisted of a web application created with HTML, CSS and JavaScript tools, a mobile application created with the Flutter framework, an application server created in the Node.js technology, while the Firebase Realtime Database was used as a database. The results of the article show the usability of the Flutter tool in real-life projects. However, the efficiency of the system components has not been tested.

For a deeper analysis of the cross-platform tools efficiency problem, it is also worth taking into account the studies on frameworks other than Flutter, as usually those technologies are also compared to native applications.

The Authors of the article [4] have analyzed the impact of using various programming tools on the performance of a mobile application. The subject of the research were the following programming tools: Android SDK with Java, Android NDK, Xamarin and Apache Cordova. For each of the platforms, an application was created that tested the execution time of the following tasks: sorting an array of 100 000 items, saving a 10 MB file to the device's storage, and reading a 10 MB file from the device's storage. However, the authors did not specify unambiguously which tool is the most efficient, because in various test cases, applications in various technologies were the fastest. The differences in execution times were determined to be imperceptible to the user. It is also important that the authors found large differences in the results between the tests carried out on the physical device and on the emulator.

The Authors of the article [5] conducted a study comparing the performance of mobile applications created with the Xamarin tool (in two versions - Forms and Native) to applications created in native technologies of Android and iOS mobile systems. The execution time was tested of applications that performed the following test scenarios: computing the number of $\pi$ to ten thousandths of a decimal place, writing and reading this number to a file, downloading a file of approximately 7 MB over the network, and reading location coordinates. In the vast majority of cases, native applications have proven to be more efficient, although these have not always been a big difference. The authors of the article concluded that the benefits of using a multi-platform tool justify its use, as long as the disadvantages of this solution are acceptable and do not have a critical impact on the project implementation.

In the article [6], the authors described the concept of creating cross-platform hybrid applications and the reason why this type of approach is present on the market. This approach was compared to two other solutions to the problem: creating native applications for each of the target mobile operating systems and creating web applications optimized for mobile devices. It has been shown that hybrid applications can combine the advantages of using one code base on many platforms, as is the case with web applications, as well as the ability to access native functions of mobile devices such as a camera or GPS by using capable API of the hardware platform.

The authors of the article [7] conducted an analysis of the possible impact of multi-platform applications on the mobile technology market. Possible advantages, disadvantages and the general impact of selecting one of the paradigms: single-platform or multi-platform on the application life cycle were shown. The analysis was carried out from the perspective of three groups related to the application: customers, programmers and suppliers of the platform on which the application runs. One of the anticipated drawbacks is the inability to fully utilize the hardware's capabilities. Flutter framework, however, differs from the solutions available at the time of writing the article, and the authors themselves promote it as a tool that allows one to achieve performance comparable to that for native applications. The authors also consider the possibility that cross-platform applications may not provide user experiences comparable to native applications.

## 3. Research method

In order to compare the time efficiency between native applications and applications developed in the Flutter framework, three pairs of Android mobile applications have been developed. Each pair of applications consists of one native application created with the Android SDK and Java language, while the other one is created with the Flutter framework. Both applications in a pair were created in such a way to be as similar as possible in terms of functionality, appearance and logic.

Each of the pairs of applications executes one of the test scenarios. The test scenarios consist of specific actions on the user interface. The work time of the threads responsible for displaying the user interface was measured, both for individual tasks and for the entire test.

The performance of the application is understood as the threads' work time, other thread states have not been taken into account. Measured time will therefore be the time of the threads' work during the execution of the task. It is worth noting that this is not the total time of the task execution, but the time during which the thread was up and running for a specific task. Such a measurement option was decided because it was the most reliable and the most platform-independent method of conducting the test.

In order to measure the work times of the threads, Android's built-in system tracing function was used. The Perfetto tool was used to trace the system. The reports generated by this tool were analyzed in an Excel spreadsheet.

Since it would be difficult to isolate the performance of individual tasks in the Perfetto report graph, 5-second breaks were introduced between individual tasks. This enables easy verification of subsequent tasks. It also has no effect on the results because the wait is not implemented in such a way that the thread is asleep, not blocked, and not doing any work. As the thread runtime is tested and not the total test execution time, the results are not negatively affected.

Research was carried out on Huawei P30 smartphone with EMUI operating system (version 12) based on Android API version 29.

### 3.1. Research scenarios

Each test scenario was carried out in following manner:

1. The researcher launches script initializing system tracing on smartphone via USB connection from computer command line.
2. The test application is launched on the mobile device.
3. The test application performs the tasks defined by the scenario. The course of the study is observed visually by the researcher.
4. After 60 seconds system tracing is stopped and Perffeto report is opened in Web browser. Report file is saved for further analysis.
5. After performing all the tests in the series, the researcher saves the relevant data from the result files to an Excel spreadsheet. The data in the sheet is analyzed and compared with the results for the second application in pair.

#### 3.1.1. Research scenario no. 1

The first research scenario involves performing operations on images. The test consists of the following tasks:

1. Application displays an image of specific size.
2. Image is replaced with different image.
3. Image is moved down. The motion is animated. Animation duration time is 2 seconds.
4. Red color filter is applied to the image.
5. Image is scaled to specific size.

#### 3.1.2. Research scenario no. 2

This scenario is based on performing actions with text characters. The test scenario includes the following tasks:

1. Text consisting of 2 000 random alphanumeric signs in a scrollable text field is displayed.
2. Font size is changed.
3. The displayed text is changed to a new random text with 10 000 random alphanumeric signs.
4. The font color is changed to green.
5. Underline is applied to the text.

#### 3.1.3. Research scenario no. 3

The last scenario includes operations on various popular user interface elements. The scenario includes the following tasks:

1. Application window with Drawer (swiping navigation menu), Appbar (bar with application title displayed on top of application) and Floating Action Bar (button displayed over application main content) is displayed.
2. 3 radio buttons are displayed in the application window.
3. One of the radio buttons is selected.
4. Radio buttons are deleted and 6 input text fields are displayed instead in the application window.
5. the text „HelloWorld!" is set in the input fields.
6. Input fields are deleted and a scrollable list of 150 elements is displayed. Each list element is a text label with a index of the element in list.
7. Scrolling the list by several items.

## 4. Results

During the tests, it turned out that while the native application performed the operations responsible for the user interface in 2 threads, as expected, the application developed with Flutter had 4 threads responsible for the user interface. The Flutter application uses both the main application thread (in the Android documentation it is called the UI thread) and the Render thread, typical for Android applications, however, the system trace output file also shows a second thread named UI and a thread named Raster. These threads are used by the Flutter system, which is confirmed by the documentation of this framework [8].

In order to distinguish between the two UI threads, the main, typical Android thread, was named the UI application thread, and the second UI thread was named Flutter's UI thread.

Since there are only 2 threads in the native application, while in the Flutter application there are 4, in order to make a fair comparison for the Flutter application, the working times of both UI threads were added together and compared to the UI thread time of the native application.

The work times of the Render and Raster Flutter threads were added and compared with the work time of Render thread of the native application.

### 4.1. Results for research scenario no. 1

As a result of the research carried out for the tests of the first research scenario, the following results were obtained:

The results for UI threads' (Table 1) show a significant difference between the execution time of the first scenario in UI threads. The native application finished work much faster.

Table 1: UI threads' work time for first research scenario

| | Flutter application | Native application |
|---|---|---|
| Task no. | Average execution time ± standard deviation (ms) | Average execution time ± standard deviation (ms) |
| 1 | 365.66 ± 32.35 | 212.22 ± 14.31 |
| 2 | 64.23 ± 6.00 | 194.01 ± 10.41 |
| 3 | 278.84 ± 8.29 | 181.86 ± 3.42 |
| 4 | 4.72 ± 0.37 | 2.25 ± 0.11 |
| 5 | 5.07 ± 0.17 | 3.72 ± 0.65 |
| Entire test | 718.53 ± 38.82 | 594.07 ± 13.35 |

The difference is especially noticeable in the first task, in the UI threads of both applications. The native application completed the task more than 100 ms faster. This is especially important because this task includes the start of the application. The difference is so large that during the observation of the course of the study, the difference in the time of starting the application was visible to the naked eye.

The second, big difference was observed in the time it takes to change the image. This time, Flutter application did the task much faster. However, when observing the examination in the Flutter application, the image "blinked" - the first image disappeared for a moment, revealing the application background below the image, and only later was the second image displayed. In the case of a native application, the image was replaced immediately, without this "blink". Thus, in terms of aesthetics, the task was performed better in the native application.

The native application performed almost all tasks faster, except for task no. 2.

Table 2: Render threads' work times for first research scenario

| | Flutter application | Native application |
|---|---|---|
| Task no. | Average execution time ± standard deviation (ms) | Average execution time ± standard deviation (ms) |
| 1 | 32.10 ± 2.01 | 55.28 ± 4.69 |
| 2 | 28.17 ± 11.24 | 49.37 ± 2.53 |
| 3 | 399.20 ± 8.12 | 334.55 ± 7.30 |
| 4 | 12.55 ± 1.17 | 10.56 ± 13.16 |
| 5 | 6.59 ± 0.92 | 4.69 ± 0.85 |
| Entire test | 478.62 ± 14.40 | 454.45 ± 14.91 |

In the case of rendering threads (Table 2), the difference between the execution times of both applications is small and in practice it is not large enough to be significant in the context of indicating a faster application.

In this case, the Flutter application performed the first 2 tasks faster, while the native application was faster from third task to the end of test.

Analyzing the results for the first of the test scenarios, it can be concluded that the main performance difference lies in the work time of the UI threads, and in particular for the task involving launching the application.

To sum up, in the first test scenario the native application turned out to be more efficient than the application created with Flutter.

### 4.2. Results for research scenario no. 2

The research for the second test scenario led to the following results:

Table 3: UI threads' work time for second research scenario

| | Flutter application | Native application |
|---|---|---|
| Task no. | Average execution time ± standard deviation (ms) | Average execution time ± standard deviation (ms) |
| 1 | 316.75 ± 8.85 | 168.09 ± 12.02 |
| 2 | 24.87 ± 4.16 | 64.11 ± 4.30 |
| 3 | 106.35 ± 6.57 | 163.48 ± 7.67 |
| 4 | 36.47 ± 9.64 | 3.18 ± 0.34 |
| 5 | 59.74 ± 18.93 | 118.31 ± 3.83 |
| Entire test | 549.97 ± 16.90 | 517.16 ± 12.15 |

The results (Table 3) indicate that once again the native application was executed more efficiently (in the context of UI threads) than the Flutter application. In the case of the second test scenario, however, the differences in the total work time of the threads were much smaller than in the first test scenario.

Detailed analysis of the table shows that in the case of text operations, some tasks were completed faster in the native application, and some in the Flutter application. Invariably, a big difference appeared when the application was started. It is worth noting that the difference in the execution times of task no. 1 for UI threads is again the largest difference among the execution times of individual tasks.

Table 4: Render threads' work times for second research scenario

| | Flutter application | Native application |
|---|---|---|
| Task no. | Average execution time ± standard deviation (ms) | Average execution time ± standard deviation (ms) |
| 1 | 34.64 ± 2.64 | 63.37 ± 2.10 |
| 2 | 20.49 ± 3.17 | 12.96 ± 3.90 |

| | | |
|---|---|---|
| 3 | 8.60 ± 6.64 | 6.44 ± 0.87 |
| 4 | 45.92 ± 4.85 | 14.98 ± 1.25 |
| 5 | 13.09 ± 3.00 | 8.04 ± 1.00 |
| Entire test | 122.74 ± 10.96 | 105.3.85 |

The work time differences for rendering threads are even lower than for UI threads (Table 4). Similar to the first test scenario, the native application turned out to be only slightly more efficient than the Flutter application.

The native application did all the tasks faster except for the first task.

The difference in the runtime of rendering threads can again be considered too small to indicate a significant performance difference to the detriment of the Flutter framework.

### 4.3. Results for research scenario no. 3

The results of the third test scenario are as follows (Table 5, Table 6):

Table 5: UI threads' work times for third research scenario

| | Flutter application | Native application |
|---|---|---|
| Task no. | Average execution time ± standard deviation (ms) | Average execution time ± standard deviation (ms) |
| 1 | 296.11 ± 9.73 | 106.98 ± 21.53 |
| 2 | 13.39 ± 2.42 | 10.74 ±0.71 |
| 3 | 31.00 ± 2.03 | 82.27 ± 5.68 |
| 4 | 53.02 ± 6.04 | 15.97 ± 0.86 |
| 5 | 68.92 ± 8.12 | 13.63 ± 1.19 |
| 6 | 50.46 ± 5.22 | 86.85 ± 6.17 |
| 7 | 20.73 ± 3.42 | 16.26 ± 1.05 |
| Entire test | 533.63 ± 18.19 | 332.83 ± 27.81 |

As in the 2 previous test scenarios, the native application again performed tasks faster for the UI thread (Table 5).

The native application performed 5 out of 7 tasks faster. The differences in the execution times of the entire test, similar to the first test scenario, are large, as the difference is over 200 ms, which, taking into account the test duration, is a very significant difference in favor of the native application.

Table 6: Render threads' work times for third research scenario

| | Flutter application | Native application |
|---|---|---|
| Task no. | Average execution time ± standard deviation (ms) | Average execution time ± standard deviation (ms) |
| 1 | 29.58 ± 2.58 | 26.09 ± 1.39 |
| 2 | 6.30 ± 0.80 | 7.36 ± 0.28 |
| 3 | 59.08 ± 4.59 | 118.85 ± 5.46 |
| 4 | 21.41 ± 2.78 | 8.03 ± 1.01 |
| 5 | 50.61 ± 2.36 | 8.42 ± 0.71 |
| 6 | 14.07 ± 1.83 | 45.30 ± 2.90 |
| 7 | 11.52 ± 1.23 | 35.66 ± 1.10 |
| Entire test | 192.59 ± 5.72 | 249.70 ± 7.64 |

Interestingly, rendering threads ran faster in the Flutter application (Table 6). This is the only case in the entire study where the average overall test execution times are lower for the Flutter application.

Moreover, it should be noted that the difference in the duration of the entire test is quite large. When it comes to specific operations, the native application performed 3 tasks faster, while the Flutter application - 4.

## 5. Conclusions

The study was successfully completed. All tests were performed and a complete set of data was obtained for analysis.

In the first test scenario, the native application definitely outperformed the Flutter application. Total thread working times were lower, and most individual tasks were completed faster by the native application. The result of the first test scenario show difference in performance of both applications. It should be emphasized that this performance difference is mainly caused by the difference in the operating times of the UI threads. Among the specific tasks, the biggest difference in execution time can be seen for task no. 1.

Much lower differences in execution times in the second research scenario. The test execution times for both the UI threads and rendering threads have little variation. The test confirms the performance differences, but in this case they were not large.

The tests of the last research scenario, like the tests of the previous scenarios, indicate a performance difference in terms of UI threads, however, for the third test scenario, the Flutter application performed faster for the rendering threads. This is the only such case in the entire study.

Summarizing the results of all tests, it can be confirmed that the native application are much more efficient in the UI thread than the Flutter application. A particularly big difference is visible when the application is started. This indicates that Flutter has to perform heavy tasks at the initiation of the application, which leads to a significant increase in the startup time of the application.

The slight performance difference was noticeable for rendering threads. Of course, in 2 out of 3 tests the native application was more efficient, but the differences were insignificant each time. Therefore, it can be concluded that Flutter's applications are satisfactorily efficient in this respect.

The research confirmed the differences in time efficiency between applications developed in the Flutter framework and mobile applications.

Android native applications are more efficient, so creating a native application will be better choice if top performance is critical aspect of application.

Considering the current state of knowledge in the area of the study, it has been confirmed again that native applications are more efficient. The study confirmed this statement for UI-based tests, which hasn't been researched before this study.

Study presents usefulness of UI-based tests in context of mobile frameworks quality evaluation.

## Literature

[1] D. Gałan, K. Fisz, P. Kopniak, A multi-criteria comparison of mobile applications built with the use of Android and Flutter Software Development Kits, Journal of Computer Sciences Institute, 19 (2021) 107-113, https://doi.org/10.35784/jcsi.2614.

[2] L. P. Barros, F. Medeiros, E. Moraes, A. F. Júnior, Analyzing the Performance of Apps Developed by using Cross-Platform and Native Technologies, International Conference on Software Engineering and Knowledge Engineering (SEKE 2020).

[3] A. M. Qadir, P. Cooper, GPS-based Mobile Cross-platform Cargo Tracking System with Web-based Application, 2020 8th International Symposium on Digital Forensics and Security (ISDFS) 2020 1-7, https://doi.org/10.1109/ISDFS49300.2020.9116336.

[4] P. Kotarski, K. Śledź, J. Smołka, Analysis of the impact of development tools used on the performance of the mobile application, Journal of Computer Sciences Institute 6 (2018) 68-72, https://doi.org/10.35784/jcsi.642.

[5] P. Grzmil, M. Skublewska-Paszkowska, E. Łukasik, J. Smołka, Performance Analysis of Native and Cross-Platform Mobile Applications, Informatyka, Automatyka, Pomiary W Gospodarce I Ochronie Środowiska 7(2) (2017) 50-53, https://doi.org/10.5604/01.3001.0010.4838.

[6] C. M. Pinto, C. Coutinho, From Native to Cross-platform Hybrid Development, 2018 International Conference on Intelligent Systems (IS) (2018) 669-676, https://doi.org/10.1109/IS.2018.8710545.

[7] L. Corral, A. Janes, T. Remencius, Potential Advantages and Disadvantages of Multiplatform Development Frameworks – A Vision on Mobile Environments, Procedia Computer Science 10 (2012) 1202-1207, https://doi.org/10.1016/j.procs.2012.06.173.

[8] Flutter framework documentation – performance monitoring, https://docs.flutter.dev/perf/ui-performance, [14.06.2022].