Performance analysis of mobile applications developed with different programming tools

Jakub Smołka[1,*], *Bartłomiej* Matacz[2], *Edyta* Łukasik[1], and *Maria* Skublewska-Paszkowska[1]

[1]Lublin University of Technology, Electrical Engineering and Computer Science Faculty, Institute of Computer Science, Nadbystrzycka 36b, 20-618 Lublin, Poland
[2] Cybercom, Hrubieszowska 2, 01-209 Warsaw, Poland

Abstract. This study examines the efficiency of certain software tasks in applications developed using three frameworks for the Android system: Android SDK, Qt and AppInventor. The results obtained using the Android SDK provided the benchmark for comparison with other frameworks. Three test applications were implemented. Each of them had the same functionality. Performance in the following aspects was tested: sorting a list of items using recursion by means of the Quicksort algorithm, access time to a location from a GPS sensor, duration time for reading the entire list of phone contacts, saving large and small files, reading large and small files, image conversion to greyscale, playback time of a music file, including the preparation time. The results of the Android SDK are good. Unexpectedly, it is not the fastest tool, but the time for performing most operations can be considered satisfactory. The Qt framework is overall about 34% faster than the Android SDK. The worst in terms of overall performance is the AppInventor: it is, on average, over 626 times slower than Android SDK.

1 Introduction

According to a recent study [1], the number of smartphone users is forecasted to grow from 2.1 billion in 2016 to 2.5 billion in 2019. This is a huge market for potential users of mobile applications. Among them, the most popular system is Android. According to a study [2], it controls the operation of 85% of all devices. Approximately 3.8 million applications can be found in the official Google Play Store [3]. These include instant messengers, media players, social networking applications, games, personal organisers and many others. All of these applications have one thing in common – they were produced using a programming tool.

There are many such tools for the Android system. These include: Android SDK (native tool provided by the creators of the system), Qt, AppInventor, PhoneGap, or Xamarin to name a few. It, therefore, seems natural to ask the question: which of these tools should be selected for application development? Will the choice reduce the functionality of the created software and will it perform quickly?

For the purpose of this study, several tests were implemented in the test applications. The purpose was to check the speed of the software produced using three tools: Android SDK, Qt and AppInventor. The first part reviews existing publications that address performance issues on Android, describe the system itself and the selected research tools. The second part discusses the prepared tests, their procedures and test environments. The third part presents the results, which are then analysed.

As already mentioned, currently the most popular system for mobile devices is Android. Its popularity is the reason for selecting it as a system in which the research was conducted. Three frameworks were selected for analysis:

- Android SDK (Android Software Development Kit) – a standard tool for creating applications for Android. It was selected to serve as a reference point for other frameworks. Java as the native language of the SDK was used.
- Qt – a popular cross-platform framework for creating applications for mobile and desktop systems. Currently supported mobile platforms include [4]: Android, Universal Windows Platform 10 and iOS. C++, as the native language of the framework, was used.
- AppInventor – an innovative development tool for creating software for Android without using classical programming languages. Applications are built using blocks, or graphic representations of the commands known from classical programming languages.

The results obtained using Android SDK are going to be used as a reference point for other frameworks. As a standard tool, it has already been used in other studies. This makes it possible to evaluate the performance of other tools which have already been or will be examined in the future. Qt was selected in order to see if its multiplatform nature incurs a performance penalty.

* Corresponding author: jakub.smolka@pollub.pl

AppInventor was selected to evaluate the consequences of using a visual-only programming method.

1.1 Related research

For mobile devices, software performance is an important parameter. It can affect not only overall user comfort, but in some cases it may also limit the functionality of the system. In [5], the authors provide a hypothetical example of an application which detects a person fall. Such functionality would be useful in particular in medical facilities such as hospitals and nursing homes. After detecting the fall, the application notifies the medical staff about the occurrence. If the application executes slowly and event detection is not immediate, the appropriateness of using such an application can be challenged. In addition, a greater execution speed also reduces energy consumption by the CPU and other phone components (energy saving mechanisms may be enabled sooner), which translates into a longer operation of the device without recharging. Therefore, it seems reasonable to address the question posed in the introduction, *i.e.*, whether and how the choice of a specific tool for software development will affect its speed.

In [6] the authors assess selected cross-platform mobile application frameworks with respect to their applications in multimedia application development. To that end, they developed two native applications (one for Android and one for iOS) and three cross-platform applications written in JavaScript using PhoneGap, Titanium and Sencha Touch frameworks. Several different aspects were compared (code length, application size, initialisation time *etc*.). The results were mixed.

In [7] a workflow for the selection of optimal parameters of C++/OpenMP code is presented. Several tests were implemented (*e.g.* sorting, matrix multiplication, searching) and their execution times for different parameter sets were evaluated. Tests were conducted on the Android platform.

In [8], H. Kang, J. Cho and H. Kim characterise AppInventor as a tool for prototyping and implementation of applications. The most distinctive feature of this framework is that it uses special graphical blocks for building the user interface and application logic. This approach to software development enables people who do not have programming experience to create their own mobile applications. A framework is not installed on the programmer's computer; it is a Web application. The finished application can be downloaded in the form of an installation package for the Android system (an .apk file).

Applications for Android written in Java are compiled into .dex files (Dalvik executable). In older versions of Android they were run on a Dalvik virtual machine [9]. However, it was possible to create applications that will run natively on the system, without involving the virtual machine or other runtime environments. It was also possible to create a program using Java language, only fragments of which would run natively. Additionally, through the use of a shared API-Renderscript, one can take advantage of the full potential offered by modern multi-core CPUs and GPUs. A. Costa and F. Almeida [9] compared the speed of execution of the native codes, Java and Renderscript. They examined the performance of different application versions in image processing (convolution function, conversion to grayscale). Renderscript was the fastest while native C was the slowest.

Code that runs on the Dalvik machine and the native code were also compared by M. C. Lin, J. H. Lin *et al.* in [10]. There were eight different types of tests: numerical calculation using recursion, use of library functions, data structures, polymorphism, nested loops, random number generation, the sieve of Eratosthenes and operations on strings of signs. The authors found a 34.2% advantage in applications using the native code. Only the generation of random numbers and heap operations performed faster on the virtual machine.

In version 5.0 of the Android system, Dalvik has been replaced by an Android Runtime (ART) virtual machine [11], which compiles the above-mentioned .dex files to the machine code during the installation process. R. Yadav and R. S. Bhadoria tested the performance of the machine using the popular Benchmark Antutu. The tests show the advantage of using the new solution, but depending on the test prevalence ranged from 1-10%. The total score achieved for Dalvik was 17,540 points and for ART was 17,857 points.

According to [12], the optimal solution is the generation of applications using a combination of both Java and the native code. However, the native method calls from Java do not perform optimally. Therefore, the authors proposed a new concept – HPO (Hybrid PolyLingual Object), aimed to improve the method calling process. They examined the time needed to call the native code from Dalvik. Using HPO yielded performance improvements in the range of 10-70%.

L. Corral, L. and G. Succi Sillitti studied in [13] the performance of the PhoneGap framework. The distinguishing feature of this tool is that it uses Web technologies for creating applications. As a result, applications can be run independently of the system platform. The following aspects were examined: accelerometer access, audio playback delay, vibration start delay, reading time of the GPS position, response time for network access, and the time of access to the phone's contacts. For the purpose of the test, two applications were implemented, one using the PhoneGap framework, and the other using the Android SDK. Only in the case of audio playback did the PhoneGap framework turn out to be 35% faster. In other tests, the Android SDK was more efficient. When acquiring the location from a GPS, it needed 500 times less time. Such large differences result from the application using another translation layer, which is the browser rendering engine.

Our literature review shows that issues of application performance on the Android platform have already been discussed (JavaScript frameworks, Native vs Java, parallel code optimisation, runtime performance, *etc*.). However, to our knowledge, there are no publications which would discuss the performance of Qt and

AppInventor. Therefore, it seemed reasonable to carry out such research in this field.

1.2 The range of tests

For the purpose of this study three test applications were implemented. Each of them has the same functionality. Performance in the following aspects was tested:

- Sorting a list of items using recursion by means of the Quicksort algorithm,
- Obtaining location from the GPS sensor,
- Reading the entire list of phone contacts,
- Saving large and small files,
- Reading large and small files,
- Converting an image to greyscale,
- Playing a sound file (including the preparation time).

The tests were implemented using functions and classes provided by the evaluated frameworks. If any functionality was not supported by a particular framework, the test for that framework was omitted. Such situations are marked as "n/a" in the presented results.

2 Environment and test procedure

To perform all the tests, the same input data were used. Each application had to sort the same list consisting of 10,000 elements. The list used in a contact data reading test contained 52 items. For each item, names and phone numbers were read. The files used in writing and reading tests were text files. In the case of small files, their size was 1000 bytes, and in the case of large files, it was 10,000,000 bytes (9.57 MB). They were read and written in the internal application folder. When writing a file, it was created entirely anew, without overwriting the existing one. To achieve the measurable value of the execution time reading and writing small files, the test operation (*i.e.* reading/writing a single file) was repeated one hundred times. For the audio playback test, a music file in mp3 format was used. The length of the audio track was one second. For the image conversion a 640 x 424 pixel test image with 24-bit colour depth was used.

To build the projects, the following versions of tools were used:

- Android SDK: Build-Tools 24.0.3, the minimum API Level required: 15, Java version: 1.8.0_111, build type: RELEASE,
- Qt: libraries - version 5.6, compiler: Android GCC arm 4.9 build type: RELEASE,
- AppInventor: MIT App Inventor 2, version: nb152a, Companion: 2.39.

2.1 Test procedure

Testing the execution times of specific functions consisted of the following steps:

1. Preparation of the test (if necessary) – loading and initialisation of all necessary variables (*e.g.* generating contents of the text files, preparing a list of items for sorting, *etc*.).
2. Reading the current system time and saving it as the start time of the test.
3. Calling the benchmarked method and saving the results of its execution, if any were returned (*e.g.* a sorted list, the status of the write operation, *etc*.).
4. Reading the current system time and saving it as the end time of the test.
5. Calculating the execution time based on the start and end times.
6. Saving the test results and displaying the execution time.

Figure 1 shows how the start and end times of the sorting test were recorded using the Android SDK framework. In this case, list initialisation, calculating the execution time and recording the result occur in separate methods.

```
public List<Integer> sort(List<Integer> list){
    super.start = System.currentTimeMillis();
    list = quickSort(list);
    super.end = System.currentTimeMillis();
    return list;
}
```

Fig. 1. Execution time measurement for the sorting test with the Android SDK.

Another example (Figure 2) shows the same measurement in the Qt framework. On this occasion, the time measurement, recording and displaying the result are performed in the same method. The procedure of time measurement remains the same. Figure 3 shows the program blocks in the AppInventor framework, which also implements the sorting test.

```
void SortTester::test()
{
    QList<int> sortedList;
    QTime start,end;
    QSettings settings;
    int execTime;
    FileManager manager(AppEngine::APP_NAME);
    start = QTime::currentTime();
    sortedList = quickSort(listToSort);
    end = QTime::currentTime();
    execTime = start.msecsTo(end);
    setLastResult(QString::number(execTime)
                  +" ms");
    manager.storeResult(TEST_ID,execTime);
    settings.setValue("lastResults/sort",
                      lastResult());
    setListToShow(convertList(sortedList));
    setListLoaded(false);
    setTested(true);
    summary->storeResult(execTime);
}
```

Fig. 2. Execution time measurement for the sorting test with the Qt framework.

Certain tests (reading location from GPS, audio playback) require framework-provided methods which run in separate threads. The difference in time measurement, in this case, is that the start time is recorded before calling the measured method, and the end time is recorded in the listener's call-back method (Java), or in a slot method connected to the signal emitted when the event occurs (Qt). The following examples (Figures 4 and 5) demonstrate how the measurement is performed in such cases.

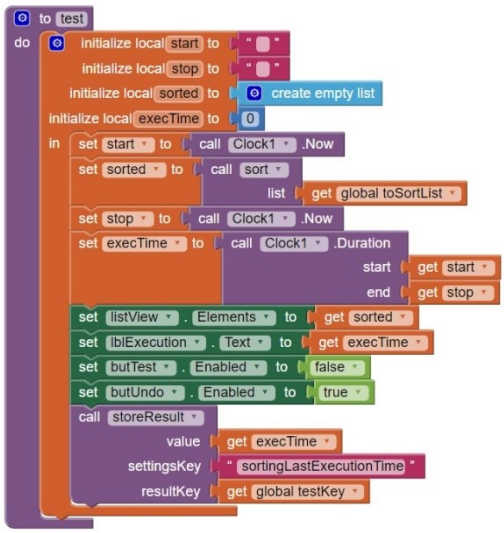

Fig. 3. Execution time measurement for the sorting test with the AppInventor framework.

```
public class SoundTester extends Tester
implements
   MediaPlayer.OnCompletionListener {
    //…
    void start(){
        super.start =
            System.currentTimeMillis();
        player.start();
    }

    @Override
    public void onCompletion(MediaPlayer mp) {
        super.end = System.currentTimeMillis();
        stopNotifyReciver.onPlayStop();
    }
}
```

Fig. 4. Execution time measurement for the audio playback test with the Android SDK.

```
SoundTester::SoundTester(…)
{
    //…
    player = new QMediaPlayer(this);
    connect(player,SIGNAL(
    mediaStatusChanged(
        QMediaPlayer::MediaStatus)),this,
        SLOT(
            onPlayerStop(
                QMediaPlayer::MediaStatus)));
}

void SoundTester::test()
{
    setTestStatus(false);
    player->setMedia(
        QUrl("qrc:/sounds/testsound.mp3"));
    start = QTime::currentTime();
    player->play();
}
void SoundTester::onPlayerStop(
    QMediaPlayer::MediaStatus status)
{
    if(status==QMediaPlayer::EndOfMedia){
        end = QTime::currentTime();
        int execTime;
        execTime = start.msecsTo(end);
        setLastTestResult(
            QString::number(execTime) + " ms");
        storeLastResult(execTime,TEST_ID,
            SETTINGS_LAST_RESULT_KEY);
        setTestStatus(true);
    }
}
```

Fig. 5. Execution time measurement for the audio playback test with the Qt framework.

2.2 Test platform

Tests were performed on a Huawei P8 Lite smartphone. Its specifications are as follows: HiSilicon Kirin 620 8 core 1.2GHz CPU, 2GB RAM, 16GB flash, Android 5.0 + 3.1 HUAWEI EMUI.

During the tests, only the test application was active on the device. All unnecessary services apart from the system services were disabled.

3 Results and discussion

Each measurement in each application was repeated 20 times. The results present the minimum, maximum and average time of 20 repetitions and are shown in Figures 6-14. The biggest performance differences were observed in the sorting test (Figure 7). AppInventor turned out to be several hundred times slower. Android SDK was ahead of Qt only in the location retrieving test and in the large file writing test (Figures 6, 12). The results are discussed in more detail in the Conclusion section.

Table 1 shows a summary of the results relative to the performance of the Android SDK. The overall result was calculated based on the average performance difference from all the tests successfully run in the framework. The Qt framework is approximately 34% faster than the Android SDK. However, it should be noted that, in the GPS location and the large file saving tests, it came out worse than Android SDK. In these cases, performance lower than 25 and 35 percent respectively was recorded. In the case of acquiring location from the GPS, Qt was also the worst among all the tested tools.

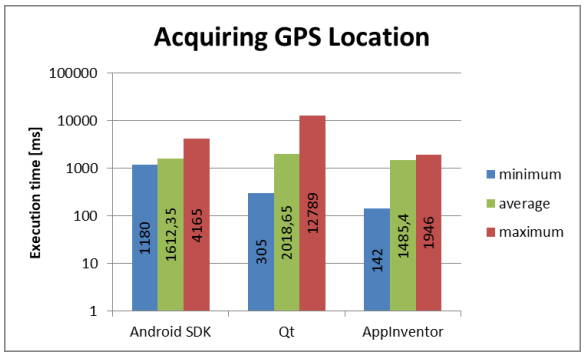

Fig. 6. Times of acquiring the location using GPS.

As expected, AppInventor was the worst in terms of overall performance. It is, on average, over 626 times slower than Android SDK. Such a bad result is mainly the consequence of the very slow execution of sorting and two file tests. Sorting a list consisting of 10,000 elements took on average 192,967 ms, which is approximately equal to 3 minutes and 22 seconds. In contrast, AppInventor achieved the best result among all the tools in the GPS and large file reading tests.

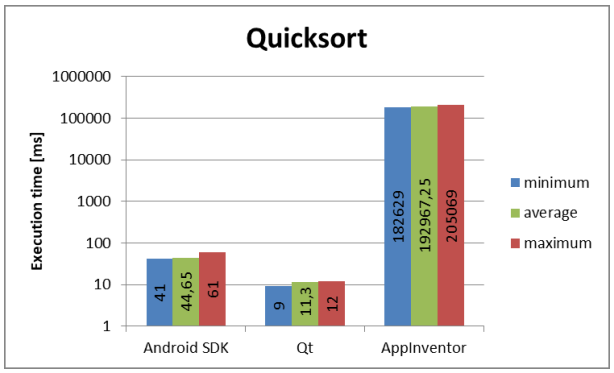

Fig. 7. Sorting times with the Quicksort algorithm.

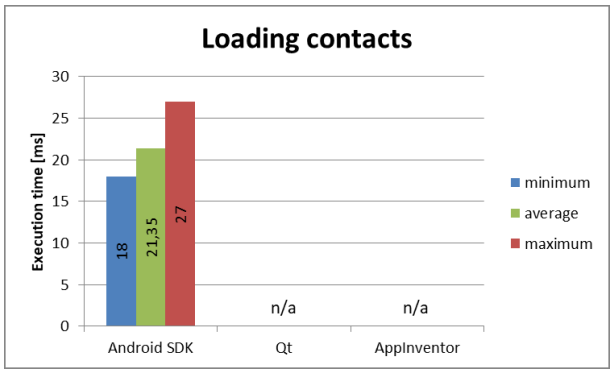

Fig. 8. Times of reading the contact list.

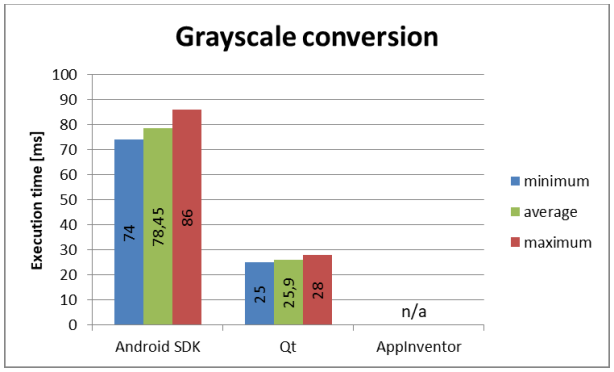

Fig. 9. Conversion to grayscale times.

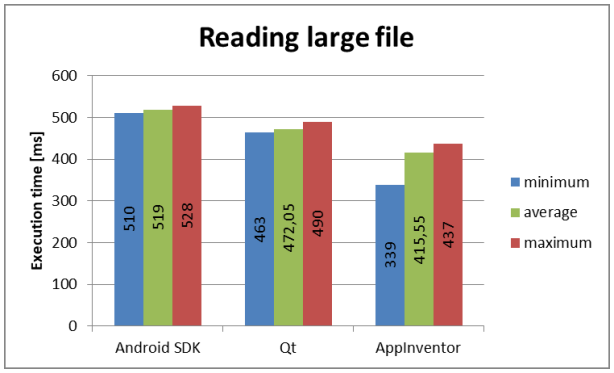

Fig. 10. Reading times of a large file.

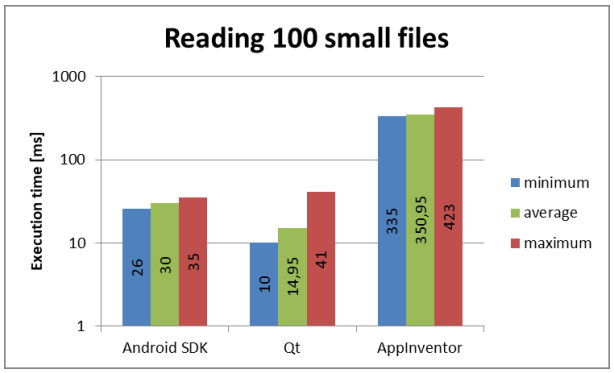

Fig. 11. Times of reading 100 small files .

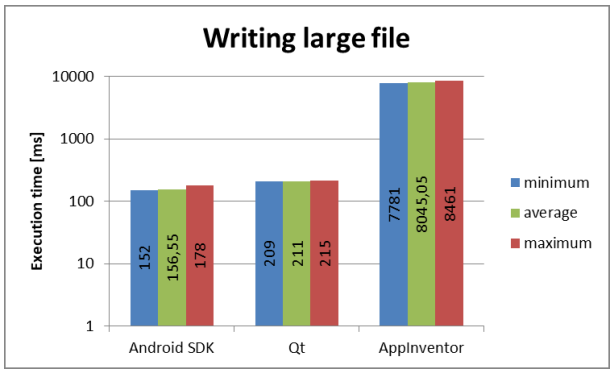

Fig. 12. Times of writing of a large file.

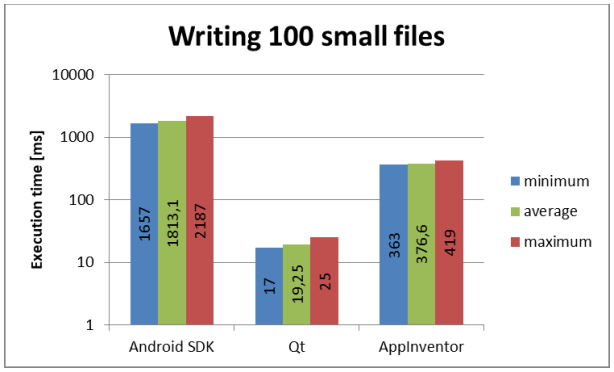

Fig. 13. Times of writing 100 small files.

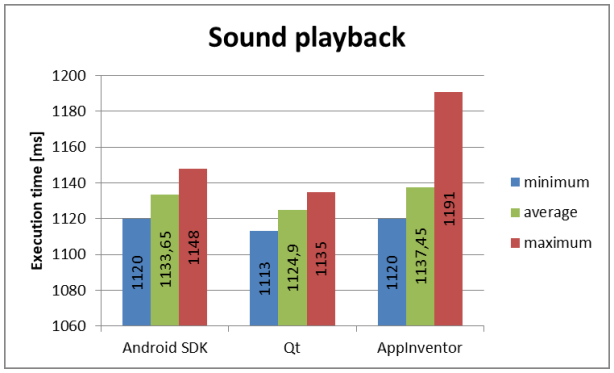

Fig. 14. Audio playback times.

Table. 1. Results of tests in relation to the Android SDK (a lower number signifies faster performance).

	Android SDK	Qt	AppInventor
GPS	1	1.25	0.92
Sorting	1	0.25	4321.77
Contacts access	1	n/a	n/a
Image conversion	1	0.33	n/a
Reading a large file	1	0.91	0.80
Reading small files	1	0.50	11.70
Writing a large file	1	1.35	51.39
Writing small files	1	0.10	0.21
Audio Playback	1	0.93	1.03
Overall	1	0.66	626.83

The Android SDK achieved satisfactory results. It is not the fastest tool, but the time for performing most operations can be considered satisfactory. Only in the writing multiple small files test are other tools clearly faster. In this case, AppInventor is almost five times faster, while Qt – a hundredfold so. The large file reading test also gives poor results, but here the differences do not exceed 20%.

It is important to note that not all of the test functions can be implemented in all of the frameworks. Despite the best performance results, Qt does not provide all the features that might be needed when creating applications on Android mobile platform. Thus, the contact reading test could not be implemented. It should be noted that accessing contacts is possible in this framework, but it requires using classes of Android SDK, which does not satisfy the assumptions of the test (using framework provided classes exclusively). AppInventor came out slightly better in this respect because the access to contacts is possible. It is provided by one of the graphical controls that can be placed in the application's GUI. Unfortunately, it does not allow one to read all the telephone contacts but only displays the list of contacts from which the user can select one item. This mode of operation prevented the implementation of any measurement. In the case of converting the image to greyscale, AppInventor turned out to be completely useless: it does not have any blocks that allow for the implementation of this task.

4 Conclusion

After a series of tests addressing the question – "which framework can produce software that runs the fastest on Android?" – the answer is surprising. One would expect that Android SDK, being the solution native to the Android platform, would be the fastest. However, among the tools tested, it is not possible to clearly indicate a framework which would be the best choice for all projects. Android SDK is admittedly slower than Qt but provides access to all the features provided by the Android system. Qt, in addition to its high speed, offers the ability to run applications on other mobile systems, which increases the number of potential users. However, the question is still open about its performance on other supported platforms.

AppInventor is a framework which does not allow one to implement functionalities that are too complicated. Its performance also leaves much to be desired. It should be noted, however, that in some respects it is superior to other tools. These are its simplicity and speed of application development. There is no need to have extensive knowledge of programming to create one's own software.

Future work may include the addition of: (1) new tests, (2) new frameworks, (3) use of non-native languages, (4) evaluation of different compilers.

In general, any programmer faced with choosing the best tools for writing applications must make the choice based on the individual application. The present study, however, should facilitate this decision.

References

1. Number of smartphone users worldwide from 2014 to 2020, http://www.statista.com [12.7.2018]
2. Smartphone OS Marker Share, http://www.idc.com [12.7.2018]
3. Number of apps available in leading app stores as of 1st quarter 2018, http://www.statista.com [9.9.2018]
4. Supported Platforms, http://doc.qt.io [12.7.2018]
5. A. Banerjee, S. Chattopadhyay, A. Roychoudhury, Advances in Computers **101**, 141-176, (2016)
6. C. M. S. Ferreira, M. J. P. Peixoto, P. A. S. Duarte, A. B. B. Torres, M. L. Silva, L. S. Rocha, W. Viana, IEEE Latin America Trans., **16**, 4, 1206-1212 (2018)
7. B. H. Phuc, P. V. Huong, N. N. Binh, L. Q. Minh, *2017 4th NAFOSTED Conference on Information and Computer Science*, 225-229 (2017)
8. H. Kang, J. Cho, H. Kim, Indian Journal of Science and Technology **8**, 19, 1-5 (2015)
9. A. Acosta, F. Almeida, Journal of Supercomputing **70**, 2, 649-659, (2014)
10. C. M. Lin, J. H. Lin, C. R. Dow, C. M. Wen, *Second International Conference on Innovations in Bio-inspired Computing and Applications*, 320-323 (2011)
11. R. Yadav, R. S. Bhadoria, *Fifth International Conference on Communication Systems and Network Technologies*, 1076-1079, (2015)
12. Y. Huang, R. Chen, J. Wei, et al., The Scientific World Journal **2014,** 1-13 (2014)
13. L. Corral, A. Sillitti, G. Succi, Procedia Computer Science **10**, 736-743, (2012)