Cross-platform Frameworks Comparison

Android Applications in a Cross-platform Environment, Xamarin Vs Flutter

Simon Hedlund   Ylva Rasmusson Wright

09-07-2021

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Swedish Title:
En Jämförelse av Cross-platform Ramverk
Android Applikationer i en Cross-plattform Miljö, Xamarin Jämfört med Flutter

Contact Information:

Author:
Simon Hedlund
e-mail: simhed98@gmail.com

Author:
Ylva Rasmusson Wright
e-mail: ylvarw@gmail.com

University advisor:
Andreas Arnesson
Department of Computer Science and Engineering

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

phone: (+46) 0455-385000
e-mail: info@bth.se
webpage: www.bth.se

Abstract

Good performance is important for an application to run smoothly for the end user, but good tools and documentation are just as important for a developer in order to be able to create good applications in the shortest amount of time.
This paper is comparing the cross-platform frameworks Flutter and Xamarin to find the respective strengths of the frameworks and which one is the better option and in what aspect, the newer Flutter or the well established Xamarin.
We did this by studying related works to the topic as well as building applications in each framework with methods to test the performance of the applications, all the while trying out the tools and documentation of each framework.
Our initial hypothesis was that Xamarin as a mature framework would perform better on average and it would also have more well developed tools. However we instead found Xamarin severely lacking compared to the newer Flutter framework and were at best equal or just slightly better.
Flutter outperformed Xamarin in CPU performance, at times 3 times better CPU performance than Xamarin, Flutter's application size being almost half of the Xamarin application and the Flutter application load times were also faster.
The tools were for most parts equal but the results of the documentations were split, with Xamarin having better component documentation with code examples for the components and Flutter having inconsistencies in documentation structure. However the Xamarin documentation was severely lacking in updated documentation and confusing instructions at places.
The only things Xamarin performed better on were the number of lines in the code as well as being marginally better performing at the RAM capacity test.
The conclusion would be that Flutter is a well performing framework that continues to develop while Xamarin feels stagnant and most of its development seems to have slowed down over the last two years.

Keywords: Android, Performance, Xamarin, Flutter, Cross-platform

Table of Contents

1 Introduction

One big problem faced by developers and companies when developing applications is the many different platforms available for the users, from smart-TV's and smartwatches to computers and smartphones.
The number of users of mobile devices have also been steadily increasing over the last few years [1].
To make the matter worse for the developers there are also different OS available for the different platforms and devices, only the mobile devices have more than one possible OS available [2] that the mobile device could be using. To develop an application specifically for each available platform, OS and device type is both costly and takes time.

In order to be able to create new products for the mobile market it is important for developers to be able to make an informed decision of what language, frameworks and other tools to use to help with building better applications and services.
One alternative is to use a cross-platform framework which not only needs a single code-base for different OS but it also allows for the application to be deployed on different kinds of devices all at once.
This allows for creation of applications from different platforms, from computers to smartphones to smartwatches with that one single code-base where only minor specifications are needed to be given for the different OS and platforms for the application to get deployed to them.

With this study we want to find out how Flutter and Xamarin, both being cross-platform frameworks, differ, what are the strengths and weaknesses and how do the end applications compare to each other in performance.
This knowledge would be useful for future development for different platforms as it is something a developer needs to keep in mind when deciding how to develop new products, as well as what tools he or she can use to reach all potential application users.
We have chosen Xamarin as one of the older frameworks with lots of time to develop [3] and to compare it with the newer framework Flutter [4].
Cross-platform frameworks is one such tool that will allow for a simpler and faster development process.
Because of the large numbers of mobile devices together with Android being the largest part of the mobile device market We have chosen to specifically compare the Android side of the frameworks.
We will accomplish this by building two as close to functionally identical applications [5, 6] as possible within the two frameworks and languages. With one application for each framework we want to see the differences in tools and performance of the two frameworks.

1.1 Background

The largest market group of devices is currently the mobile device market [7] and of these the majority of the devices is using the Android OS with Android representing a majority of the market share [8] and more than 70% [2] of the mobile market belongs to Android based devices.
For this reason it's highly relevant to compare tools and framework for Android application development.
This is why this thesis will be focusing on Android applications as it is highly relevant for developers to be able to select good tools for Android development.
Another reason to focus only on Android applications is because of the limitations of time and resources available for this thesis.

There are a number of cross-platform frameworks available for a developer to choose from when creating an application, but in order to get full access to all the device built-in features there is not much that can compare to building an native application.
However, it´s not always the case that a developer wants to write directly in native language as it takes time to build native applications for the different OS and it's often much easier and quicker to build an application with a cross-platform framework to support you with building the native applications.
With this in mind, during this thesis we will compare two different cross-platform frameworks for building Android applications, Flutter and Xamarin [9, 10].
An older established framework like Xamarin [3] with a newer, more modern one like Flutter [4]. Both frameworks compile to native Android code, ARM and x86 libraries [11, 12] when building a mobile application but since they are using different compilation methods they are highly interesting to compare.

We have chosen Xamarin because it is an well known older framework that appeared at the beginning of the rise of smartphones [13], it has been around for a while and have had a long time to develop [14].
At the other hand Flutter is a newer framework created in a time where smartphones and mobile devices are commonplace. This has led to a framework developed while fully aware of the prevalence of mobile devices today [15].
Another interesting point is that the size of the user base of these two frameworks were quite small in 2019, but over the last year the difference in usage between the frameworks have become quite prominent [16]. Where if we look at the statistics of the user base of the frameworks Flutter went from 30% of developers using it in 2019 to 39% in 2020, during the same time Xamarin went from 26% in 2019 to 14% of developers using Xamarin.
As we can see Flutter is gaining a lot of popularity and users while Xamarin is loosing a lot of users.

So with all of this in mind we want to see if this change in user base is representative of one framework performing better or if it's simply a trend.
This is vital for developers to know when deciding on a framework in the long term as a declining user base might simply be a sign of a change in trending framework or it

might indicate a declining framework where other better alternatives have appeared. This is what we intend to look at, is one framework better than the other in application development and performance for Android applications? Do they perform equally well or if one is better than the other? And based on our findings, which framework is the recommended one to use?

1.2 Purpose

The aim of this paper is to show the pros and cons of developing in either Xamarin or Flutter from a mostly performance related standpoint with a focus on the Android applications.
With the experiments we are going to perform we will get statistics of the performance of the Flutter and Xamarin Android applications.
This information will be valuable for both developers and companies looking to develop performance demanding applications for mobile devices but still want to use a cross platform solution.
In addition to our performance related work we will also look at the tools available for these frameworks as performance doesn't matter if developers can't work efficiently in the language and framework specific tools and components.
If this wasn't the case everyone would still be working in assembly to work as close to the machine as possible.
Hopefully this study can also be of help to independent developers and students who have an interest in mobile application development and performance as well as for developers that have specific performance needs for an application.

1.3 Scope

This thesis will analyze and compare the performance as well as developer tools for the android application in the cross-platform frameworks Xamarin and Flutter during the development if an application using each framework. These applications in both Flutter an Xamarin will have performance tests made in them and we will do an analysis of the code metrics on the completed applications.
Because of previously mentioned (1.1) resource limitations as well as the market sizes of devices, platforms and OS we will develop these apps towards the Android platform for mobile devices and focus our tests on the completed android applications.
The applications performance testing will include application startup time, RAM and CPU tests as well as metrics such as code complexity, size of the completed applications and lines of executed code from the completed applications.

We will also compare the documentations to each other as well as the tools used during development and for running performance tests. We will end the thesis with a conclusion about what cross-platform framework, if any, is better during different use cases as well as an overall recommendation if possible.

1.4 Glossary

Cross-Platform framework:	A framework to deploy code to different OS and devices
OS:	Operating System
Dart:	Coding language
C#:	Coding language
Flutter:	Cross platform framework using Dart
Xamarin:	Cross platform framework using C#
Android:	OS for mobile devices
CPU:	Central Processing Unit
RAM:	Random Access Memory
AVD:	Android Virtual Device
Logcat:	Application log

1.5 Objectives

This project will consist of a comparison of two cross-platform frameworks, Flutter and Xamarin and the focus will be on the Android application side of the frameworks. In order to compare the frameworks and the resulting android applications we will be looking at some relevant points for comparison of the two frameworks.

- See how demanding the mobile framework applications are on the end users mobile phone.
  This is important for the usability for the end users. If an application noticeably slows down the phone then the user is more likely to either not use the product or to simply uninstall the application [17]. This is why we have decided to measure performance of the frameworks

- Find out what tools and support can a developer expect in the different frameworks.
  We will be utilising some of the tools for the frameworks during the development and performance testing of the applications and during this we'll be evaluating the usability of the tools. We will be looking at where we encounter problems during the development, the severity of said problems as well as how easily they were resolved

- To find out when a developer should choose a specific mobile framework and why, what are the strong and weak points of the different frameworks.
  This will be decided by the performance of the applications as well as comparing the results from previous mentioned comparison points.

2 Objective

2.1 Research Questions

RQ1

How do documentation, tools and tool usage differ? We will look at similarities and differences in tools for development in the frameworks during the application development. How does the documentation compare as well as the support for when problems arise during application development.

RQ2

What are the strengths and weaknesses of the completed Android applications? We will compare lines of code of the applications, the size of the finished applications as well as the code complexity of the source code.

RQ3

How do the mobile frameworks compare in performance? The performance we will look at is the application startup time, CPU usage as well as RAM usage.

2.2 Research Questions Explained

RQ1 As a developer you are dependent on the tools for the framework in order to execute and test the code, if the tools can not be used as intended the work becomes that much harder and more complicated and time consuming to do for the developers.
The same applies to the Documentation which is one of the main tools for a developer to learn how to use a framework [18].
As such it is an important aspect of development using a framework since an developer might choose a different framework to use if the first framework is unusable because of bad documentation and tools [19].

RQ2 The question of code complexity and lines of code can be an important point for developers to know when they choose their language as this can give an overview over the time a project might take. Both of these metrics is an indication of code quality since more complex code and bigger source code generally takes more time to develop and is harder to maintain [20, 21].
Application size can also be a deciding factor as having a Large application is costly for the end user, it takes time and bandwidth to download, and it takes up space on the device [22].
A large application size might also give the end user an impression that an application

is badly programmed or is bloated with unnecessary content/features that will simply take up unnecessary space on the users device [23]. This question will be answered through an experiment.

RQ3 Performance is important for developers to take into consideration when building applications that are in need of lots of computations or need to perform multiple tasks since if an application needs a long time to buffer the user will quickly lose interest in the product.
As such it's good for a developer to know if a framework can handle performance well.
As these are important factors to consider when developing an app we are going to test CPU [24] and RAM [25] usage of the frameworks applications as well as the startup time [26] of the applications. This question will also be answered through an experiment.

Experiments Motivation When it comes to selecting your framework of choice a lot of factors have to be considered. In this paper we want to focus on the factor of performance. With this we want to find out which framework handles the different performance criteria better.
With Xamarin being an older framework developed over a long time it would be interesting to compare that to a new framework like Flutter.
If the application performs badly in CPU and RAM usage the end user experience will suffer [27, 28, 29].
Having long startup times and large file sizes can be detrimental to an application's success as these can annoy the end user [17].
while building the applications for the experiment we also want to look at the code complexity and lines of code since these statistics are interesting for the developer to know before using a framework or language [20, 21].
But having good documentation is also a very important aspect of development for a programmer as it will show what functionality the framework offers and how to implement them. Good documentation is something that will save the developer lots of time otherwise spent on research or trial and error [18].
Having good tools is another such functionality which will immensely help the developers to create the applications as they support the development process [30].

Experiments Expectation For file size we are expecting the completed Flutter application to have a smaller file size. this is because of when comparing the base example "Hello World" application in Flutter where it at the smallest.
Flutter reaches 4.7 MB [31, 32] compared to Xamarin's base "Hello World" application which reaches a full 15.8 MB [33]. We are expecting this difference in size to also transfer to the completed applications as well.
We are expecting to see Xamarin with its mature Components to come out on top in terms of CPU and RAM usage. With the time Xamarin has had to develop their performance optimization during compile and run time it is probable that Xamarin will perform better in performance related tasks.

As both C# and Dart are based on C style syntax [34, 35] we expect the lines of code as well as the code complexity to be roughly the same. This could turn out to be false as it depends on how different the frameworks are from the original languages syntax wise.
We are also expecting Xamarin to have a more comprehensible and in-depth documentation build up over the years as well as more tools for easier development, yet again since that is something that is expected with a large user base and time to develop the tools in the first place [36].

3 Research Method

3.1 Related Works

We will perform a study of related works in this paper. We will be studying relevant literature to find out more about Dart, Flutter, Xamarin and C# in relation to performance.
The main search phrases used to find literature and papers were: Flutter, Xamarin, performance, mobile application, cross-platform and Android.
The primary sources for finding publications about the subjects have been:

- BTH Summon
- BTH DIVA
- Google Scholar

In the case of use of white papers and gray literature we will cross reference the content with other sources in order to keep a high level of accuracy of the content.

3.2 Application Experiment

For this paper we will be building two applications, one in each framework and using each frameworks tools [5, 6].
We will be building similar applications in each language with the same content, the methods will be as similar as each language/framework permits but they will execute the same commands and type of calculations.
During the building of the applications and tests we will not be using multi-threading since Flutter is single threaded [37].
We will be using the application in order to measure CPU, RAM , application size, startup time and code metrics [25, 24, 20, 38, 21]. As such the applications will contain features that are demanding for both CPU and RAM.
The applications will be run on an android emulator running a Google Pixel 2 with android Android 9, API level 28. All tests will be run from the same computer [Annexes Table 3].

With the performance of the computer we will be using for the tests [Annexes Table 3], the emulator will not run into performance issues.
The computer is using an Intel processor which is superior when it comes to smartphone emulations [39].
The reason to run an application in an emulator is because this will give a fresh system with no background processes or damaged hardware. This will in turn make for better statistics while doing some pretty heavy performance testing using an emulator allows us to not risk doing damage to an actual device.
Another reason to use an emulator is to make the repeatable by both ourselves as well as others as well as bring consistency in the tests, and even if the applications will be run with the use of an emulator it will not impact the result in any way [40] since all tests on both applications will be done in the exact same environment.

Using an emulator, the applications will be able to run with no background applications running so it won't be disturbed by other applications using up Ram or CPU power while the tests are running and to minimize external differences between test runs.
For styling and routing we will use the preset you get when creating the application as these are not essential for the running of the tests, as such we kept the preset minimal design.
We will also create splash screens to make it closer to a real application startup. This splash screen will consist of minimal content of an icon and a background colour.

The tools we will use to do the performance tests are as following:

- Android studio [41]
- Android studio Logcat [42]
- Android studio Emulator [43]
- Flutter [9]
- Dart Stopwatch [44]
- Visual studio 2019 Enterprise [45]
- Visual studio 2019 Enterprise Logcat [46]
- Visual studio 2019 Enterprise Emulator [47]
- Xamarin [10]
- C# Stopwatch [48]

In the case of the CPU usage we will make a method to find prime numbers which is one well established method to do benchmarking tests [24]. We will measure CPU for 5000, 10000, 50000 and 100000 calculations respectively. For a statistically accurate measurement we will be running the test 1000 times for each calculation level and take the average of these runs as the results.

For the RAM capacity test we will create a list to be filled with objects through a loop [25]. As the loop goes around the RAM will be filled up with objects and the application will eventually crash.
With each iteration of the loop we will have a counter that will increase with every object added to the list.This test will be run 5 times for each application to get an accurate result.

As the tools or IDE we are using gets updates and new versions get released during the application development we will use the new versions. If something was to break because of these updates we will roll back to the working version of the

framework or IDE.
We will use this approach since it's a more realistic way of the application build process, since as a developer you always want to update your tools as security risks and extra features get patched or added.

Versions of Tools and Frameworks

Xamarin Xamarin.Forms version 5.0 release 3

Flutter Version 2.0.3

Visual Studio 2019 Enterprise edition, Version 16.8.7

Android Studio Version 4.1

Dart Version 2.12.1

Stopwatch Dart No separate version to class

Stopwatch C# No separate version to class

Android studio: AVD Version 30.4.5

Android studio: logcat integrated part of Android Studio, no separate version

Visual Studio 2019: AVD integrated part of Visual Studio, no separate version

Visual Studio 2019: logcat integrated part of Visual Studio, no separate version

Any release of new framework versions happened after the application was already completed and not during the development process and did as such not affect the development.
Updates that did occur:

Visual Studio 2019 Enterprise new version 19.9.4, several updates occurred, no effect on development process.

Android Studio No new version occurred during the development of the Flutter application. During testing: New major version 4.2, release date: 2021-05-04, This would have impacted the tests with updates to profiling tools. All tests were run with the new version.

3.3 Statistics Gathering From Completed Applications

When gathering data of the CPU test, we will measure the time it takes for each framework to complete these calculations. This will be done by using the Stopwatch class in each of the frameworks.
The stopwatch will start before the calculations are started and end when the calculations are done. We will then collect the results from the stopwatch by returning the results to the console log.

The RAM capacity will be measured by populating a list with as many objects as possible.
When the app crashes we will print out a counter of the number of objects to see how many objects the application was able to create.
For startup time, application size and code metrics no coding is needed but some tools are needed to measure these statistics.
For startup time measurement we use logcat as logcat logs the startup time on applications started on android devices. We start each app 5 times and take the average value of those start times.

For application size we check the registered size on the phone emulator in the phone settings. As these are quite small projects the vast majority of the size will come from sources we have no control over as it is a part of the framework build.

Lastly we have code metrics where we have different solutions for the two frameworks.
In Xamarin we use Visual Studio's already implemented tool for code metrics [49].
In Flutter we use a Flutter CLI [50] package called Flutter_code_metrics [51]. The data we gather from these tools will be used to measure code complexity.

The reason for the different approaches is because Android studio does not have built in tools to measure code complexity in Flutter projects. As code complexity is calculated in the same way the different approaches will not affect the results.

When measuring code complexity and lines of code we will compare the functions and methods in the application. We will not compare files that we did not interact with.
To avoid getting a bloated lines of code count we will exclude comments, we will also be following the language syntax for the frameworks [52, 53].
The complexity of the example below will be equal as well as the line count won't count the opening brackets as a separate line so even if we use the recommended syntax for each language this won't impact the number of executable lines of code.

The bracket placement we use in respective application in accordance to each language style [54, 55] however these differences in code style will be taken into consideration when counting lines of cod so the difference will not impact the result. An example of what the difference in code style for bracket placement looks like:

Dart

```
if(something){
    do code;
}
```

C#

```
if(something)
{
    do code;
}
```

4 Related Works

We have been looking at related works about the topics of Flutter, Xamarin, Dart and C# as well as cross-platform frameworks.
There were surprisingly few papers about the subject of the Flutter framework and Dart even though it have been around for a few years [4] already and is created and advertised by Google [9] as such one would think that such a framework would be quite well known with an amount of studies relative to the amount of users [56]. The same was surprising with Xamarin where there were quite a lot of books about the framework but not a lot of papers that were less than 5 years old.
We were expecting to see more papers with positive results for Xamarin since it has been able to survive as a framework for a long time but the results were unexpected with more positive results for Flutter than for Xamarin.

The study of Andersson Vestman and Karlsson 2018, comparing Xamarin Android applications to an Native Android application concluded that Xamarin has worse performance than a native Android application.
When they were running performance tests in the form of iterative Bubble sort [57] and Fibonacci [58] sequences they concluded that the performance of RAM and CPU for Xamarin based applications were much slower than native Android applications, with the best Xamarin result being 10.4% slower and at worst a whole 77.9% slower than a Native Android application [59] .

Another Paper comparing Xamarin and native development by Borop 2018, Where they compare Xamarin.Forms to native applications, Borop finds in the paper that Xamarin were surprisingly efficient to work with and in the example of buttons in Android vs Xamarin the method in Xamarin is much more efficient to use and were quicker to build applications with.
However, Borop also notes the differences that can appear when the code is translated to native languages, in the paper he mentions how the shape of the buttons got interpreted in different ways between the platforms [60].

A paper from 2020 by Olsson, is comparing performance and looks of Flutter and Native applications. The paper concludes that the performance of Flutter is comparable to Native applications when looking at CPU usage and lines of code.
For the Flutter application the lines of code count were even lower than for the native applications (Flutter:125 lines, Android: 217 lines and IOS: 363 lines) [61].

Another study by Stender and Åkesson 2020, writes about comparing Flutter and React Native and they conclude that Flutter have somewhat better CPU usage than React Native (2% lower CPU usage for Flutter) and they conclude that there were a ”slight advantage to Flutter in overall performance” [62].

In the Paper from Biørn-Hansen, A., Rieger, C., Grønli, TM. et al 2020, they Compare different kinds of cross-platform methods, among others compilation-Based which is the method used by Flutter and the runtime-based and Interpreted type such as Xamarin, though in this paper they use NativeScript which also is a runtime-

based and Interpreted framework, the paper were showing some interesting inherent differences between the different approach for the frameworks.
They mentions among other "other cross-platform frameworks were found to possibly be more performant on certain metrics, such as NativeScript's time-to-completion and CPU usage, and Flutter's minimal increase in memory usage during task benchmarking".
They conclude in their findings that when using a cross-platform framework you might get lower performance than an native application, but depending on the selected metric compared and framework you might get close or equal performance as well which shows the importance of selecting Framework after what performance needs a project might have [63].

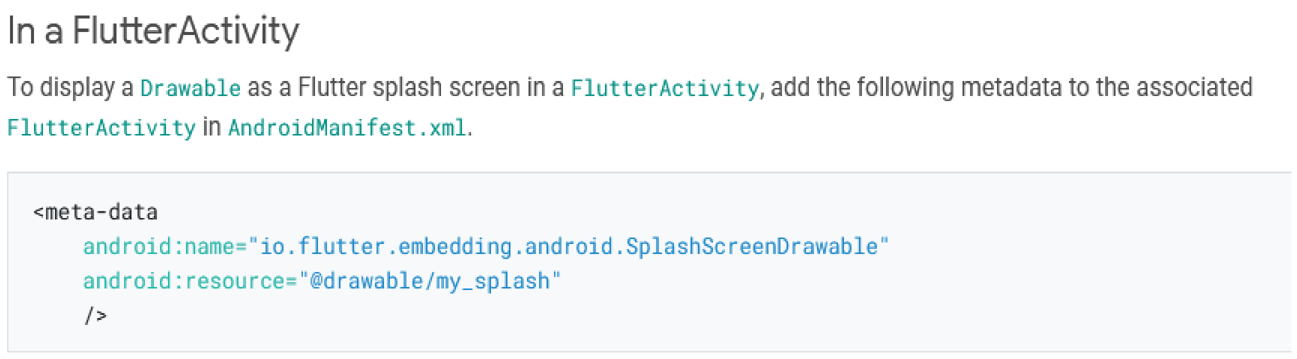
5 Results

5.1 RQ1

Documentation

The documentation of the two frameworks were generally quite extensive, however some points stand out from them that were noticed.
Flutter had some inconsistencies in the way the documentation were written, in the example below both fig.1 and fig.2 are from the documentation of a splash screen [64], where it sometimes mentions in what file the code is in and sometimes not.
Other times the Flutter documentation is missing examples and only shows the general usage of an component(fig.3) [65]. When it comes to Xamarin the documentation

had a good coverage of the framework and it were especially useful with the companion repository on GitHub with examples for each framework component [66] which is very useful to see how a component is implemented and used.
However, Xamarin also has the two biggest disadvantages of the two frameworks. The first one is the old documentation, some parts have not been updated for years, one such part is the Xamarin Profiler documentation that at the moment got updated 2018.
Another big drawback is that since the documentation is sometimes not 100% accurate, like in the case of Xamarin Profiler, the UI of the actual Profiler and the images in the documentation is not all the same (Annexes fig.5-7). But also some misguiding instructions that shouldn't be there in the first place (Annexes fig.8).
These aspects of the Xamarin documentation makes it hard for the developers to use newer tools and also makes it hard to know if the documentation still is relevant.

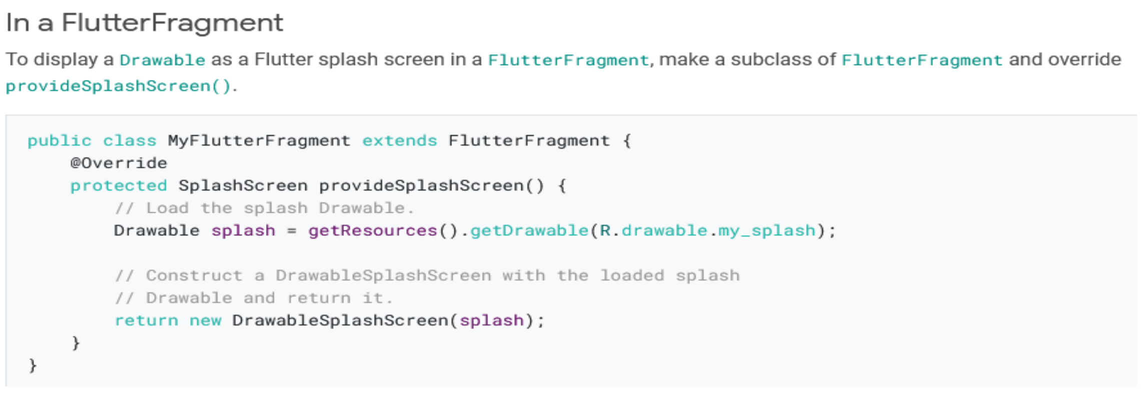


In a FlutterActivity

To display a Drawable as a Flutter splash screen in a FlutterActivity, add the following metadata to the associated FlutterActivity in AndroidManifest.xml.

```
<meta-data
    android:name="io.flutter.embedding.android.SplashScreenDrawable"
    android:resource="@drawable/my_splash"
    />
```

fig 1. Flutter documentation with files specified



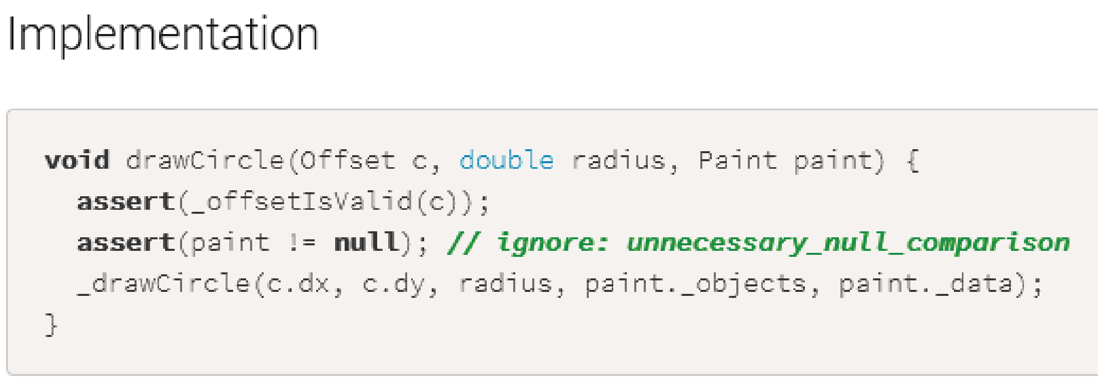
In a FlutterFragment

To display a `Drawable` as a Flutter splash screen in a `FlutterFragment`, make a subclass of `FlutterFragment` and override `provideSplashScreen()`.

```
public class MyFlutterFragment extends FlutterFragment {
    @Override
    protected SplashScreen provideSplashScreen() {
        // Load the splash Drawable.
        Drawable splash = getResources().getDrawable(R.drawable.my_splash);

        // Construct a DrawableSplashScreen with the loaded splash
        // Drawable and return it.
        return new DrawableSplashScreen(splash);
    }
}
```

fig 2. Flutter documentation without files specified



Implementation

```
void drawCircle(Offset c, double radius, Paint paint) {
  assert(_offsetIsValid(c));
  assert(paint != null); // ignore: unnecessary_null_comparison
  _drawCircle(c.dx, c.dy, radius, paint._objects, paint._data);
}
```

fig 3. Flutter documentation without full example

Tools

Android virtual device were used in both Visual Studio as well as in Android Studio. we could use the same AVD in both text editors without having to create separate AVD's and could install both applications in one AVD, this was very useful when we tried to eliminate all external factors when running the tests as well as saves a lot of space on a developers computer as one AVD is able to take up several GB of space on a computer.

Stopwatch were used to measure the time the calculations took in the methods of the CPU tests. Both Dart and C# had each their version of the tool and we found almost no difference in either the implementation or the function of the tools.

Logcat is the application logging tool in both Android Studio and Visual Studio Enterprise, and here too we could not find any distinguishable differences in usability or functionality.

Visual Studio 2019 is the recommended text editor for developing Xamarin applications and they are both made by Microsoft[10].
However, one drawback with Visual Studio is that a developer interested in being able to run a profiler tool to get an basic insight in an application's performance is forced to use the subscription service Enterprise version as the profiler is not a

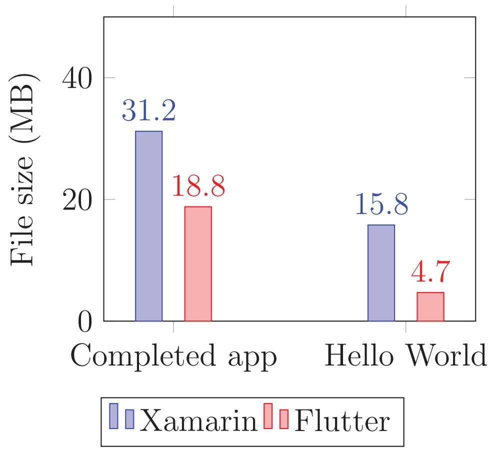
part of the free version of Visual Studio 2019, Visual Studio 2019 Community, and the profiler also needs to be installed separately from Visual Studio 2019 which adds extra setup steps in order to be able to profile the applications.
Visual Studio 2019 has its own SDK-manager and AVD-manager which makes it easy to develop and test android applications. But the basic AVD-manager in Visual Studio had fewer controls of the AVD than in Android studio, it only had controls to create and delete an AVD, no wipe data or cold boot options.

Android Studio were used for the development of the Flutter application with the Flutter and Dart Plugins, this worked really well for development and there were no problems with the development of the Flutter application.
Android Studio also has its own AVD-manager as well as SDK-manager for Android application development. However in Android Studio the profiler is built in already as one of the essential tools for basic performance checks of an application.
It was also quicker to develop the application in Android studio (as can be seen in the time to develop the applications) as the application didn't need to go through a solutions build first before building to the android device.
Instead the Flutter application could be directly built to the Android device as it built and compiled at the same time, saving the developer time otherwise used to wait for build to finish.

5.2 RQ2

File Size

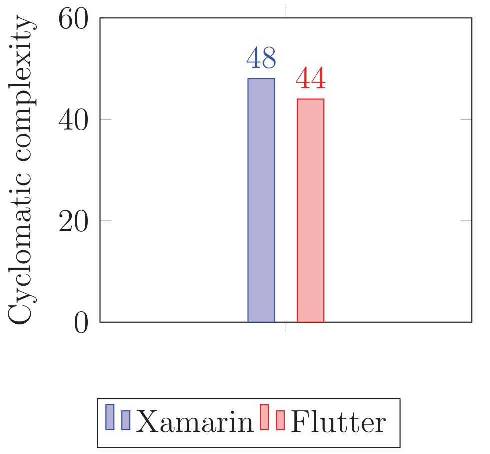
When it comes to file size we installed the completed applications in release mode on the android emulator and measured the different file sizes on the applications. Our Flutter application grew from 4,7 MB to 18.8 MB while Xamarin grew from 15.8 MB to 31.2 MB. Flutter grew by 14.1 MB while Xamarin grew by 15.4 MB.



Graph 1. File size of the completed applications and Hello World applications.

Cyclomatic Complexity

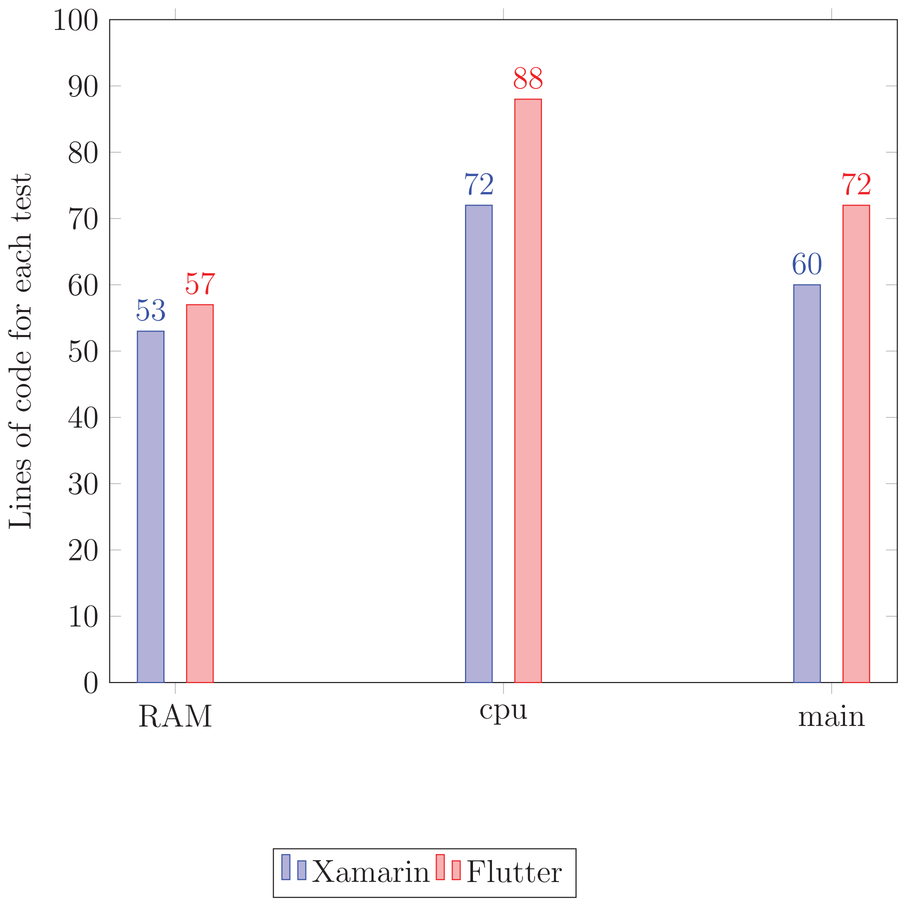
The cyclomatic complexity was almost equal between the two applications. With both languages being based on C syntax [34, 35] this is to be expected as the syntax will be similar.



Graph 2. Cyclomatic complexity of the code.

Lines of Code

In lines of code Xamarin is clearly more compact. Xamarin files are divided into .xaml pages for styling and .xaml.cs files for logic. On the other hand Flutter uses the same file for logic and styling.
The main pages are not identical which will affect lines of code.
We use the default front pages which is why they are slightly different.
The routing is also different, in Xamarin we use a burger menu while in Flutter we use buttons for navigation.
The CPU and RAM test are as close as equal that we could get with the differences in the frameworks and languages structure.

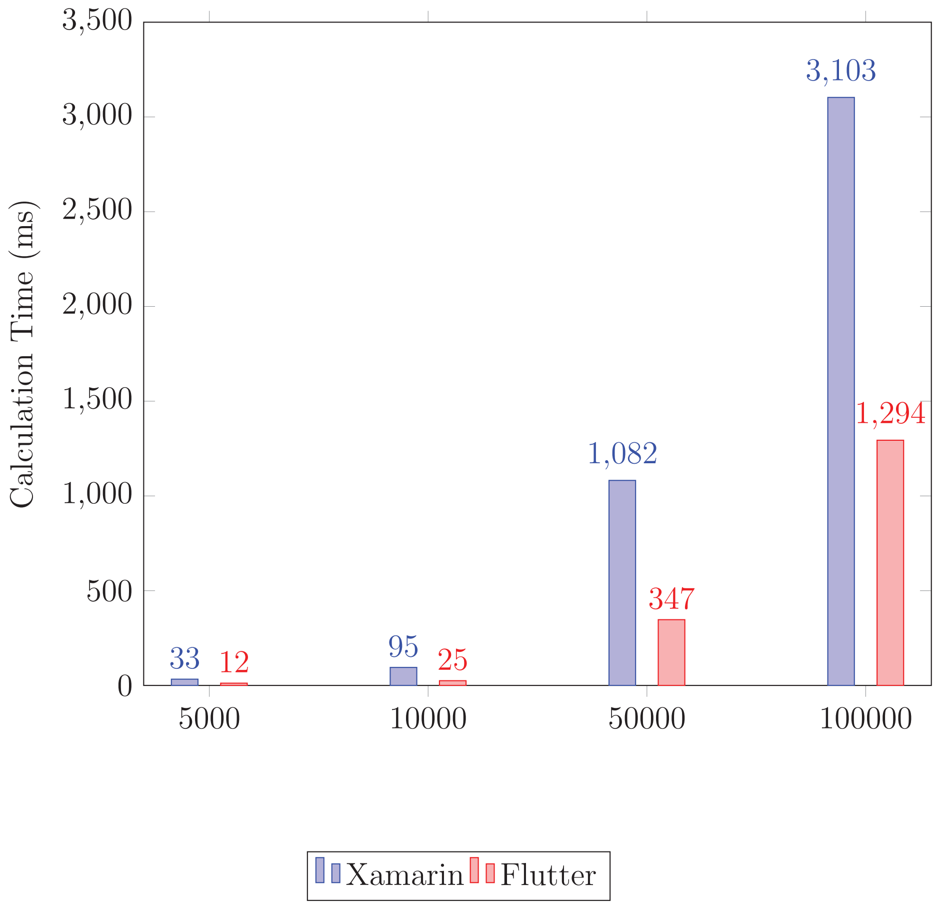


Graph 3. Lines of code used for the different tests.

5.3 RQ3

CPU Tests

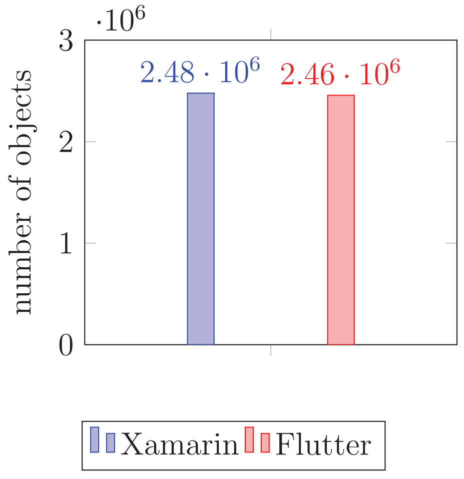
In graph 4 we can see that Flutter is faster in all our tests. For most calculation levels Flutter is roughly 3 times faster than Xamarin.



Graph 4. Calculation time for prime numbers n times.

RAM Tests

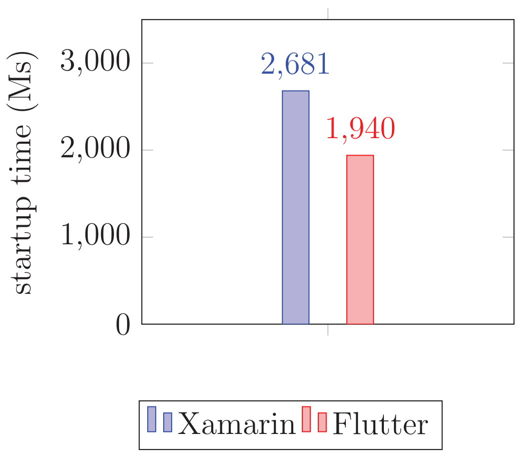
As seen in graph 5, both frameworks are able to create close to the same amount of objects in The RAM test. We can see that Flutter and Xamarin are almost equal, with Xamarin being capable of slightly more objects in memory than Flutter.



Graph 5. number of objects created.

Application Startup Time

The applications each have a simple splash screen during startup implemented in accordance to each framework Documentation, both containing a simple icon and a background colour.
Each application was launched 5 times an the average time of those times was our final startup time logcat shows us that when it comes to startup time Flutter is the winner by about 0.7 seconds.



Graph 6. Application startup time.

6 Analysis & Discussion

6.1 RQ1, How do documentation, tools and tool usage differ?

The tools we have been using from the two frameworks are the tools focused on measuring and developing/building the code and application.
The tools for measuring the time of the calculations were very similar between both languages as well as the usage which also were very similar between them.
The use of the IDE's were at most times the same as well. Though one of the disadvantages for Xamarin is the by Microsoft recommended, text-editor Visual Studio 2019, that in of itself is not a problem but it does not have an pre-installed profiler tool.

In order to run the Profiler tool for an application during the development you will need to install the tool separately as well as to subscribe to Visual Studio 2019 Enterprise [67, 68]. In comparison to using the free version of Visual Studio Community or as for Flutter where it's very easy to run the application in Android Studio and its profiling tools which are installed as a part of the IDE.
Outside of that there are not many differences between the available tools that were found and the ones tested during the application development were implemented and functioned almost exactly the same between the frameworks.

The Android Virtual Device and SDK-manager were almost the same, but the AVD-manager in Android Studio had more functionality and were thus more effective to work with, especially with the cold boot and wipe data options the Android studio AVD-manager have.

Both logcat and Stopwatch had close to identical functionality and the ways to implement Stopwatch were also identical so there were no differences to compare in the case of these tools.
The biggest difference between the frameworks outside of the text editors used were the Documentation of the two frameworks.
The Xamarin Documentation [69] were very comprehensive with a companion example repository in GitHub [66].
The examples were very helpful when building components and features and is also a good support for new developers who might otherwise be insecure of how to write the code.

When using the Flutter documentation [70] it was not as comprehensible, it didn't have any example code as extensive as Xamarin have and it is a bit inconsistent with how they describes usage of methods. Sometimes the Flutter documentation mentions the specific file a piece of code needs to be in, sometimes they simply mentions how the code needs to be structured but not in what specific file it needs to be in.
A good example of this were the splash screen where the documentation didn't mention in what specific file the splash interface needed to be in [64].

However, the biggest demerit is still to Xamarin where, even though it have an extensive documentation, some parts the documentation haven't been updated in several years [67].
This gives the developers a huge disadvantage as the try to use new techniques but can't find support in the documentation for implementation.
A documentation page with several years since the last update makes a developer doubt the credibility and usefulness of the documentation as a tool to support development of an application. This is because it is hard for a developer to know if the documentation has been neglected or if the documentation is still relevant and the content simply hasn't changed over this time.

6.2 RQ2, What are the strengths and weaknesses of the completed Android applications?

When it comes to lines of code Xamarin wins with having the fewer lines in all of the files (graph 3). This could be because of the way Xamarin is structured to split the styling into another file.
The styling syntax is also very different between the two frameworks while the logic is almost identical. Styling is still important as we can't use the applications and use the test code logic without it.
This is why the styling is included in the lines of code.

The size of the finished applications is a metric that matters for the user since they have a limited amount of memory available on the device.
The size of the finished applications differed quite a lot in the basic "Hello World" applications with Xamarin being almost 3 times larger than Flutter in size in the basic applications and this difference was sustained to the finished applications as well (graph 1).
Although the difference had become somewhat less prominent in the completed applications the Xamarin application were at the end still almost double the size of the Flutter application and this discrepancy in application size will probably continue for a while as the applications grow larger in size.
So with this result we were correct in the early estimates that Flutter would be the more memory efficient application. This is something that will impact the users of the finished application and it will have an effect on the customer satisfaction if the users have to download large applications to their devices.

The complexity of the source code in the applications are very similar and as both languages are based on C syntax and it's not surprising to see that the applications have a similar cyclomatic complexity (graph 2).
Though they were similar there were some differences in the cyclomatic complexity between Flutter and Xamarin.
The cyclomatic complexity for the applications is a bit lower in the Flutter application. The C# code was a bit longer and needed more checks in it which quickly makes the language have a higher cyclomatic value which makes it harder to use and read [21].

This difference in complexity was easily felt when comparing the code of the CPU tests between Dart and Flutter, where the Dart code is easier to quickly get a grasp of its methods.

6.3 RQ3, How Do The Mobile Frameworks Compare In Performance?

The results from the performance tests have been quite surprising at times with the biggest surprise being the large difference in results of the CPU tests.
As shown in graph 4 we can see that for all the intervals of calculations we tested Flutter only needed about 1/3 of the time of what Xamarin needed in order to do the calculations for the prime numbers.
This seems to scale linearly which would mean that Xamarin could become unbearably slow quite quickly in CPU intensive tasks. This quite large discrepancy between the framework's test result might be attributed to the different methods the applications are translated to native code as one is compilation based (Flutter) and another is run-time based (Xamarin).
As previously mentioned in the related works this difference in performance result is also supported by Biørn-Hansen et al [63] where their comparison of different methods of compilation also found different results based on what compilation method was used.

As in the paper from Andersson Vestman and Karlsson [59] comparing RAM and CPU in Xamarin to Native application where Xamarin performed poorly at best, this compared to in Olsson's comparison of Flutter CPU usage versus Native application [61] where Flutter performed really well. This supports the findings in the tests made that Flutter should perform better than Xamarin.

When looking at the startup time it was about the expected amount of time to start up an application, however we weren't expecting such a big difference in the startup time between Flutter and Xamarin.

When launching the applications the difference in startup speed was a whole 0.7 seconds. Both applications launch in a reasonable time, being under 3 seconds for each (graph 6) [26].
We do not know how well launch time scales with larger applications so further testing would be needed for a conclusion on whether the difference would scale with the application size.

We also saw some differences between the frameworks in how efficient they were with the RAM memory use however these differences were minimal being less than a 1% difference (graph 5).
Both of the applications seem to be able to handle more than enough for most projects, and the biggest difference in results would depend more on the hardware of the user than the RAM capacity of the frameworks.

But as previously mentioned among the related works by Biørn-Hansen et al [63], the Flutter platform and the Xamarin platform is using different approaches in its architecture to build the applications on the different platforms [71, 12], as such you could expect to see some differences also in the performance between the frameworks.

But looking at the general results from the related works, the Flutter framework seems to perform better than Xamarin when comparing to native applications as well as when they are compared to other frameworks such as React Native [62, 61, 59].
Only one of the papers we looked at found Xamarin to be efficient and this were the results of Borop comparing Xamarin to native development [60].

7 Conclusion

During this project our goal has been to make a comparison of the frameworks to find out in what ways they differ and what strengths and weaknesses they have when comparing the two.

The Documentation for the frameworks both had their weaknesses as well as strong points. But the biggest difference was the severity between the respective weaknesses, Flutter had some inconsistencies in the structure of the documentation pages with what format information was written in, which is still acceptable.
The Documentation for Xamarin had a much more severe weakness of not being regularly updated as some parts were several years old as with the profiling tool and the actual profiler had a slightly different UI than the one shown in the documentation as well as misgiving information (fig 4-7). At the other hand, one of the biggest strengths of the Xamarin documentation was the Companion repository with all the component implementation examples. But over all the Flutter tools and components felt easier to use since the Documentation were more accurate, up to date as well as that Android Studio were ready to run Flutter with very little effort of installations.

When comparing the number of lines of code Xamarin had fewer lines, partially due to the code structure of the frameworks, which made Flutter to have more executable lines of code, Flutter didn't separate Style from methods either which made the general size of the individual files larger compared to Xamarin's refactored structure where the style were separated from the logic.
Flutter had a smaller application size compared to Xamarin. The size difference was about 40% in the end and while both apps were small in our tests, if this trend continues into bigger projects Xamarin projects could become huge very quickly.
When it comes to cyclomatic complexity the values were very similar with Flutter winning out by a small margin. When it comes to cyclomatic complexity there is just a minor difference which did not affect development in any meaningful way.

Comparing the performance of the CPU usage, Flutter performed around 3 times better than Xamarin during our CPU tests. This was consistent during all testing intervals made. With this huge difference we recommend using Flutter for anything CPU intensive as a Xamarin application would be much slower than a Flutter application. In our RAM test the applications had almost identical statistics leading to it not really mattering what framework. The results were very good for both frameworks so RAM usage should not be a problem for the end application. The Flutter application performed better by 0.7 seconds on startup time compared to the Xamarin application. As both apps launched in less then 3 seconds we don't see any problems in startup time as 3 seconds is still very good.

Overall Flutter has proven to have good performance and outperformed Xamarin in almost every aspect except for RAM usage where Xamarin performed slightly better than Flutter as well as that Xamarin had a slightly lower lines of code count. Flutter outperformed Xamarin in most tests as well as the tools and documentation. Flutter both looks and feels like a modern framework and the time to set up as well as the time it took to get the application ready for testing were all much shorter than the Xamarin application. But when it comes to what framework a developer should choose between Flutter and Xamarin, in almost every aspect it would be recommended to choose Flutter over Xamarin.

8 Validity Threats

limited test runs The first risk to the results is the number of tests done during the test phase of this project.
With more tests done we would have gotten more statistically secure results and trends and tendencies would have been more clear.
It would also be interesting to gather the range of each test run with the lowest and highest value for each run to compare the difference in range.

limited types of performance tests There are many different points of comparison one is able to use when measuring performance. Had other types of performance tests been used the result of the comparison between the frameworks might have turned out differently.

Number of devices used Since the tests were all run in the same type of AVD we can not make conclusions of the frameworks performance in relations to different hardware in the device as well as if the Android version on the device would give different results, this thesis can only test the frameworks in relation to each other.

9 Future Works

In Depth Performance Tests: As there are other performance criteria one can test the full performance test scope might change the overall outcome as the frameworks might perform differently in different tests.
It Would also be interesting to do more types of performance tests such as object rendering, page loading speed as well as testing the device hardware functionality through different frameworks.
And as we were not using multi threading during the tests it would be interesting to make comparisons with multi threading as well and see how the frameworks compare then. However, as flutter is not able to use multi-threading [37] it would be a comparison of Flutter without or with alternative solution in the form of packages [72] and compare with Xamarin with and/or without multi-threading. The results could possibly change in favor Xamarin then.
It would also be possible to recreate the tests as is but with the addition of IOS to compare the frameworks to both the IOS and Android OS mobile device applications.

Native or React Native Comparison: It would be very interesting to compare Flutter and Xamarin with React Native as well as that is another well known, modern cross-platform framework with many users. The same applies to comparing the two frameworks Android application not only to each other but also to a Native application to see the difference against the Native performance.

Comparison of multiple OS and devices: In this thesis we focused on only the Android smartphone applications. But with more time and resources it could be valuable with a more extensive comparison of the two frameworks as performance might differ between platforms and OS.

Comparison of different application sizes: As in the case of startup time the results might vary depending on the application size so this would be interesting to compare.

References

[1] S. O'Dea. *Number of smartphone user growth.* URL: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. published: 25-06-2021, accessed: 17-02-2021.

[2] StatCounter. *Mobile Operating System Market Share Worldwide.* URL: https://gs.statcounter.com/os-market-share/mobile/worldwide. published: 2021, accessed: 17-02-2021.

[3] Miguel de Icaza. *Announcing Xamarin.* URL: https://tirania.org/blog/archive/2011/May-16.html. published: 2011, accessed: 17-02-2021.

[4] Flutter. *Flutter releases.* URL: https://flutter.dev/docs/development/tools/sdk/releases. updated: 2021, accessed: 17-02-2021.

[5] Y. Rasmusson Wright S. Hedlund. *Repository for Flutter application.* URL: https://github.com/corgisout/exjobb_flutter. published: 2021, accessed: 15-05-2021.

[6] Y. Rasmusson Wright S. Hedlund. *Repository for Xamarin application.* URL: https://github.com/ylvarw/xamarinTestApp. published: 2021, accessed: 15-05-2021.

[7] StatCounter. *Desktop vs Mobile vs Tablet Market Share Worldwide.* URL: https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet. published: 2021, accessed: 17-05-2021.

[8] Statista. *Market share forecast for smartphones operating systems.* URL: https://www.statista.com/statistics/272307/market-share-forecast-for-smartphone-operating-systems/. published: 2019, accessed: 17-02-2021.

[9] Flutter. *Flutter - beautiful native apps.* URL: https://flutter.dev/. published: N/a, accessed: 03-03-2021.

[10] Microsoft. *Xamarin.* URL: https://dotnet.microsoft.com/apps/xamarin. published: N/a, accessed: 18-05-2021.

[11] M. Goderbauer Et al S. Zakhour P. Chalin. *How does Flutter run my code on Android?* URL: https://flutter.dev/docs/resources/faq#run-android. updated: 04-05-2021, accessed: 02-03-2021.

[12] C. Dunn Et al D. Ortinau D. Britch. *Architecture.* URL: https://docs.microsoft.com/en-us/xamarin/android/internals/architecture. published: 25-04-2018, accessed: 02-03-2021.

[13] P. Svensson. *Smartphones now outsell dumb phones.* URL: https://web.archive.org/web/20130801114353/http://www.3news.co.nz/Smartphones-now-outsell-dumb-phones/tabid/412/articleID/295878/Default.aspx. published: 29-04-2013, accessed: 17-02-2021.

[14] Altexsoft. *The Good and The Bad of Xamarin Mobile Development.* URL: https://www.altexsoft.com/blog/mobile/pros-and-cons-of-xamarin-vs-native/. updated: 13-11-2020, accessed: 31-05-2021.

[15] Y. Luchaninov. *Why Flutter is a Silver Bullet of Cross-Platform App Development.* URL: https://mobidev.biz/blog/flutter-app-development. published: 10-06-2020, accessed: 31-05-2021.

[16] S. Liu. *Cross-platform mobile frameworks used by software developers worldwide in 2019 and 2020.* URL: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/. published: 02-07-2020, accessed: 16-05-2021.

[17] Dot Com Infoway. *Why users uninstall your app.* URL: https://www.dotcominfoway.com/blog/infographic-why-users-uninstall-your-app/. published: 2017, accessed: 05-03-2021.

[18] M. Nielson. *Why Documentation Matters.* URL: https://medium.com/@oswebguy/why-documentation-matters-7152d46448e1. published: 26-07-2017, accessed: 2021-06-08.

[19] D. Adalaide P. Ghaffari. *What factors do you consider when deciding which framework to use when developing a new application?* URL: https://www.quora.com/What-factors-do-you-consider-when-deciding-which-framework-to-use-when-developing-a-new-application. published: 2014, accessed: 16-06-2021.

[20] Sonam Bhatia and Jyoteesh Malhotra. '"A survey on impact of lines of code on software complexity"'. In: *2014 International Conference on Advances in Engineering Technology Research (ICAETR - 2014).* 2014, pp. 1–4. DOI: 10.1109/ICAETR.2014.7012875.

[21] Christof Ebert et al. 'Cyclomatic Complexity'. In: *IEEE Software* 33.6 (2016), pp. 27–29. DOI: 10.1109/MS.2016.147.

[22] Gonçalo Alvarez. *"App size matters I".* URL: https://www.farfetchtechblog.com/en/blog/post/app-size-matters-i/. published: 18-05-2020, accessed: 07-06-2021.

[23] P. Reinhardt. *Effect of Mobile App Size on Downloads.* URL: https://segment.com/blog/mobile-app-size-effect-on-downloads/. published: 05-10-2016, accessed: 08-06-2021.

[24] PassMark Software. *CPU Benchmarks.* URL: https://www.cpubenchmark.net/cpu_test_info.html. published: N/a, accessed: 16-05-2021.

[25] E. Berg D. Wallin. *Microbenchmark.* URL: http://user.it.uu.se/~danw/darkII/stridelab.html. published: 2001, accessed: 18-05-2021.

[26] Colin Mo. *Why Should You Care About Your Mobile App's Startup Time?* URL: `https://blog.embrace.io/why-should-you-care-about-your-mobile-apps-startup-time/`. published: 16-04-2021, accessed: 14-06-2021.

[27] AppDynamics. *Mobile app performance explained.* URL: `https://www.appdynamics.com/media/uploaded-files/mobileapp.pdf`. published: 2014, accessed: 05-07-2021.

[28] R. O'Connor. *Why Mobile App Performance Matters.* URL: `https://www.progress.com/blogs/why-mobile-app-performance-matters`. published: 11-12-2015, accessed: 05-07-2021.

[29] Reza Rawassizadeh. 'Mobile Application Benchmarking Based on the Resource Usage Monitoring'. In: *IJMCMC* 1 (Oct. 2009), pp. 64–75. DOI: `10.4018/jmcmc.2009072805`.

[30] R. Sheldon. *The right mobile app dev tools matter.* URL: `https://searchmobilecomputing.techtarget.com/feature/The-right-mobile-app-dev-tools-matter`. published: 29-07-2016, accessed: 05-07-2021.

[31] altexsoft. *pros and cons of flutter app development.* URL: `https://www.altexsoft.com/blog/engineering/pros-and-cons-of-flutter-app-development/`. published: 2021, accessed: 15-02-2021.

[32] M. Goderbauer Et al S. Zakhour P. Chalin. *How big is the Flutter engine?* URL: `https://flutter.dev/docs/resources/faq#how-big-is-the-flutter-engine`. updated: 22-02-2021, accessed: 06-03-2021.

[33] microsoft. *Release Packages.* URL: `https://docs.microsoft.com/en-us/xamarin/android/deploy-test/app-package-size#release-packages`. published: 2018, accessed: 06-03-2021.

[34] Upwork Staff. *C vs. C++: Which Language is Right for Your Software Project?s.* URL: `https://www.upwork.com/resources/c-sharp-vs-c-plus-plus`. published: 2019, accessed: 19-01-2021.

[35] D. Bolton. *The Fall and Rise of Dart, Google's 'JavaScript Killer'.* URL: `https://insights.dice.com/2019/03/27/fall-rise-dart-google-javascript-killer/`. published: 27-04-2019, accessed: 19-01-2021.

[36] Reinhold Plösch, Andreas Dautovic and Matthias Saft. 'The Value of Software Documentation Quality'. In: (2014), pp. 333–342. DOI: `10.1109/QSIC.2014.22`.

[37] O. Brand. *Flutter Threading.* URL: `https://medium.com/flutter-community/flutter-threading-5c3a7b0c065f`. published: 20-12-2018, accessed: 08-06-2021.

[38] Sealights. *Code Quality Metrics: Is Your Code Any Good?* URL: `https://www.sealights.io/code-quality/code-quality-metrics-is-your-code-any-good/`. published: N/a, accessed: 08-06-2021.

[39] D. Ortinau et al J. Douglas J. Montemagno. *Hardware acceleration for emulator performance (Hyper-V HAXM)*. URL: https://docs.microsoft.com/en-us/xamarin/android/get-started/installation/android-emulator/hardware-acceleration?pivots=windows. updated: 13-02-2020, accessed: 10-06-2021.

[40] P. Krishnakumar A. Pai. *Testing on Emulators vs Simulators vs Real Devices*. URL: https://www.browserstack.com/guide/testing-on-emulators-simulators-real-devices-comparison. published: 20-06-2019, accessed: 09-06-2021.

[41] Android Studio. *Android studio*. URL: https://developer.android.com/studio. updated: 2021, accessed: 29-06-2021.

[42] Android. *Measure app performance with Android Profiler*. URL: https://developer.android.com/studio/profile/android-profiler. published: 2021, accessed: 03-03-2021.

[43] Android dev. *Run apps on the Android Emulator*. URL: https://developer.android.com/studio/run/emulator. updated: 2021, accessed: 29-06-2021.

[44] Flutter. *Stopwatch class*. URL: https://api.flutter.dev/flutter/dart-core/Stopwatch-class.html. updated: 2021, accessed: 10-05-2021.

[45] Microsoft. *Downloads*. URL: https://visualstudio.microsoft.com/downloads/. published: N/a, accessed: 17-02-2021.

[46] N. Schonning et al D. Ortinau C. Dunn. *Android Debug Log*. URL: https://docs.microsoft.com/en-us/xamarin/android/deploy-test/debugging/android-debug-log?tabs=windows. updated: 22-06-2018, accessed: 14-06-2021.

[47] Microsoft. *Visual Studio Emulator for Android*. URL: https://visualstudio.microsoft.com/vs/msft-android-emulator/. updated: N/a, accessed: 29-06-2021.

[48] Microsoft. *Stopwatch class*. URL: https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.stopwatch?view=net-5.0. published: N/a, accessed: 10-05-2021.

[49] J. Kirsch Et al M. Jones G. Hogenson. *Code metrics values*. URL: https://docs.microsoft.com/en-us/visualstudio/code-quality/code-metrics-values?view=vs-2019. updated: 02-11-2018, accessed: 17-02-2021.

[50] Ashe, K. Walrath R. McGinnis and J. Sharma. *flutter: The Flutter command-line tool*. URL: https://flutter.dev/docs/reference/flutter-cli. updated: 05-02-2021, accessed: 17-02-2021.

[51] D. Krutskikh. *dart code metrics 3.2.2*. URL: https://pub.dev/packages/dart_code_metrics. updated: 2021, accessed: 17-02-2021.

[52] B. Wagner. *General Structure of a C Program (C Programming Guide)*. URL: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/general-structure-of-a-csharp-program. published: 14-05-2021, accessed: 18-05-2021.

[53] S. Zakhour Et al K. Walrath P. Chalin. *Language samples.* URL: https://dart.dev/samples. updated: 09-03-2021, accessed: 18-05-2021.

[54] B.Wagner. *General Structure of a C# Program.* URL: https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/program-structure/. updated: 14-05-2021, accessed: 09-06-2021.

[55] P. Chalin Et al B. Nystrom K Walrath. *Effective Dart: Style.* URL: https://dart.dev/guides/language/effective-dart/style. updated: 30-06-2021, accessed: 09-06-2021.

[56] Stack Overflow. *technology-most-loved-dreaded-and-wanted-languages-loved.* URL: https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-languages-loved. published: 2020, accessed: 19-01-2021.

[57] Wikipedia. *Bubble Sort.* URL: https://en.wikipedia.org/wiki/Bubble_sort. updated: 10-06-2021, accessed: 05-05-2021.

[58] Wikipedia. *Fibonacci number.* URL: https://en.wikipedia.org/wiki/Fibonacci_number. updated: 28-06-2021, accessed: 05-05-2021.

[59] M. Karlsson F. Andersson Vestman. 'Experimentell studie av prestandaskillnader mellan native Android och Xamarin för mobilapplikationer'. In: (2018).

[60] A. Borop. 'Xamarin Forms vs Native Platform Development'. In: *Student Scholarship – Computer Science. 5.* (2018). DOI: https://digitalcommons.olivet.edu/csis_stsc/5.

[61] M. Olsson. 'A Comparison of Performance and Looks Between Flutter and Native Applications: When to prefer Flutter over native in mobile application development'. In: (2020). DOI: http://urn.kb.se/resolve?urn=urn:nbn:se:bth-19712.

[62] H. Åkesson S. Stender. 'Cross-platform Framework Comparison: Flutter React Native'. In: (2020). DOI: http://urn.kb.se/resolve?urn=urn:nbn:se:bth-19749.

[63] TM. Grønli et al A. Biørn-Hansen C.Rieger. 'An empirical investigation of performance overhead in cross-platform mobile development frameworks'. In: *Empir Software Eng 25* (2020). DOI: https://doi.org/10.1007/s10664-020-09827-6.

[64] A. Rajkumar Et al S. Zakhour T. Sneath. *Adding a splash screen to your mobile app.* URL: https://flutter.dev/docs/development/ui/advanced/splash-screen. updated: 2021, accessed: 13-05-2021.

[65] Flutter. *drawCircle method.* URL: https://api.flutter.dev/flutter/dart-ui/Canvas/drawCircle.html. updated: 2021, accessed: 18-05-2021.

[66] J. Swan. *Xamarin Example Code.* URL: https://github.com/xamarin/xamarin-forms-samples. updated: 13-02-2021, accessed: 22-02-2021.

[67] C. Dunn Et al D. Ortinau D. Britch. *Xamarin Profiler.* URL: https://docs.microsoft.com/en-us/xamarin/tools/profiler/?tabs=windows. updated: 03-06-2018, accessed: 10-05-2021.

[68] Microsoft. *Compare Visual Studio Products*. URL: `https://visualstudio.microsoft.com/vs/compare/`. published: N/a, accessed: 15-05-2021.

[69] Microsoft. *Xamarin Documentation*. URL: `https://docs.microsoft.com/en-us/xamarin/`. published: N/a, accessed: 22-02-2021.

[70] P. Chalin Et al S. Zakhour K. Walrath. *Flutter Documentation*. URL: `https://flutter.dev/docs`. updated: 01-04-2021, accessed: 22-02-2021.

[71] H. Muller et al T. Sneath M. Beltran. *Flutter architectural overview*. URL: `https://flutter.dev/docs/resources/architectural-overview`. updated: 16-06-2021, accessed: 18-05-2021.

[72] A. Mezoni. *Threading*. URL: `https://pub.dev/packages/threading`. published: 27-07-2018, accessed: 08-06-2021.

Annexes

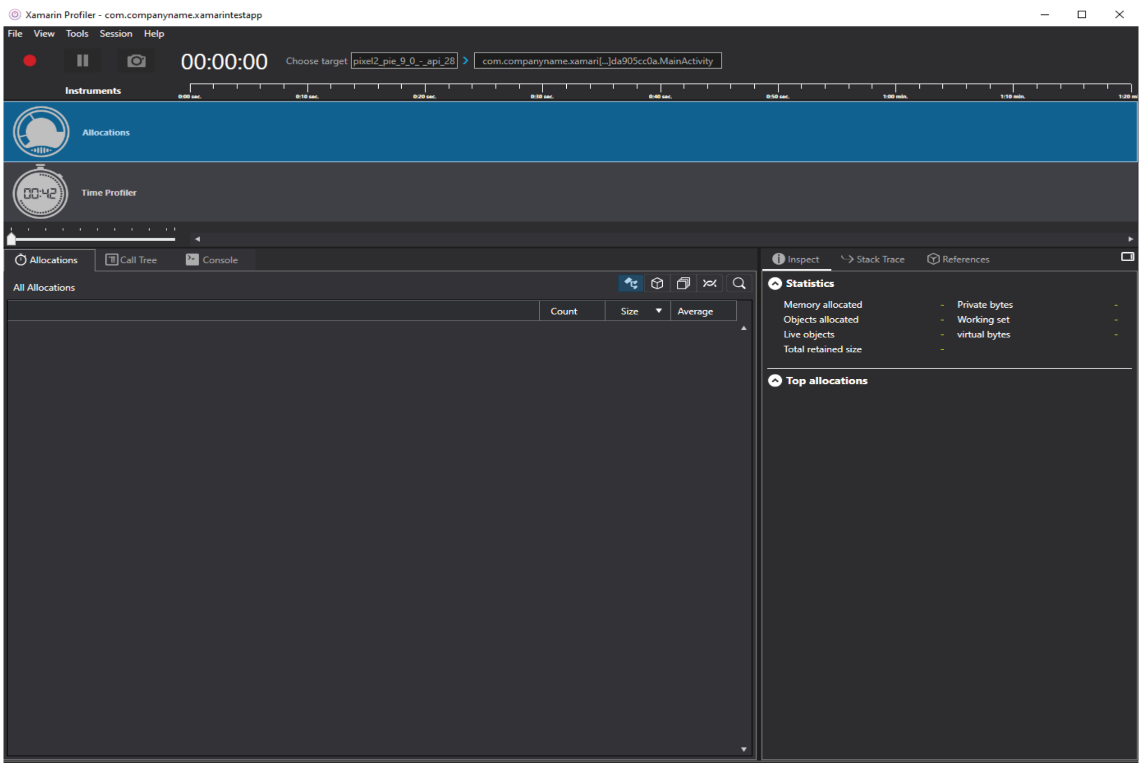

Xamarin Profiler - com.companyname.xamarintestapp
File View Tools Session Help
00:00:00
Choose target pixel2_pie_9_0_-_api_28 com.companyname.xamari[...]da905cc0a.MainActivity
Instruments
Allocations
Time Profiler
Allocations Call Tree Console
All Allocations
Count Size Average
Inspect Stack Trace References
Statistics
Memory allocated -
Objects allocated -
Live objects -
Total retained size -
Private bytes -
Working set -
virtual bytes -
Top allocations

fig 5. The actual Xamarin Profiler after installation

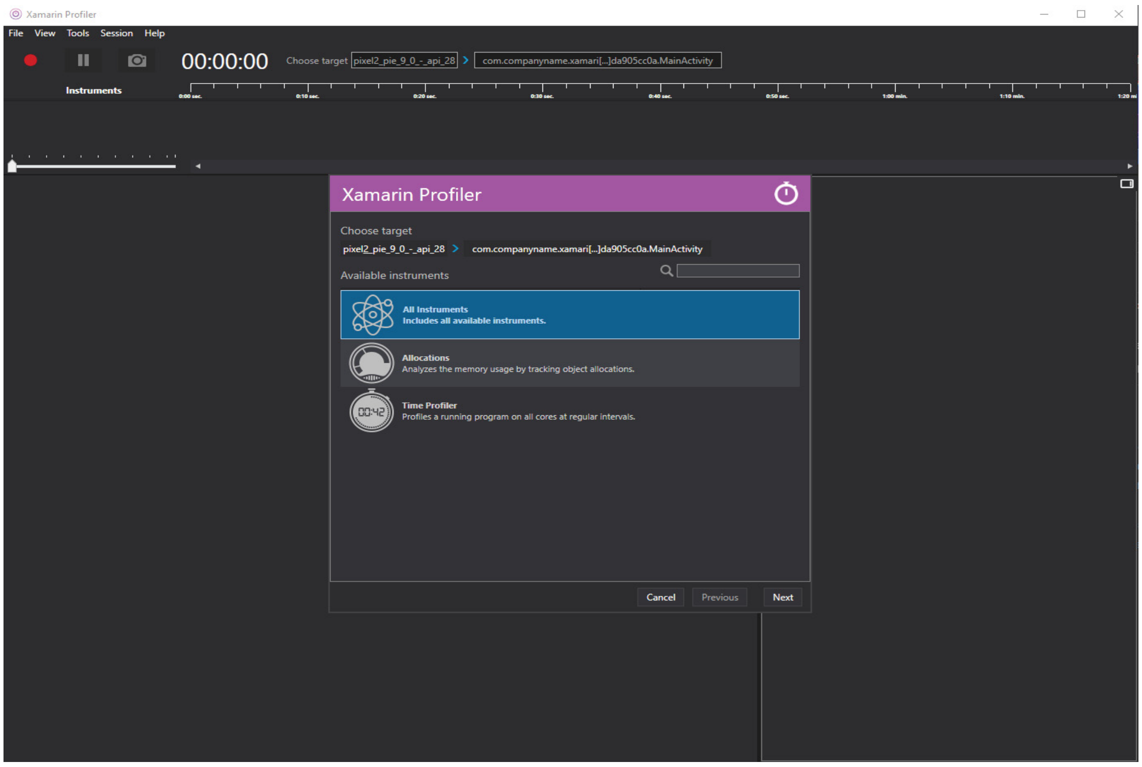
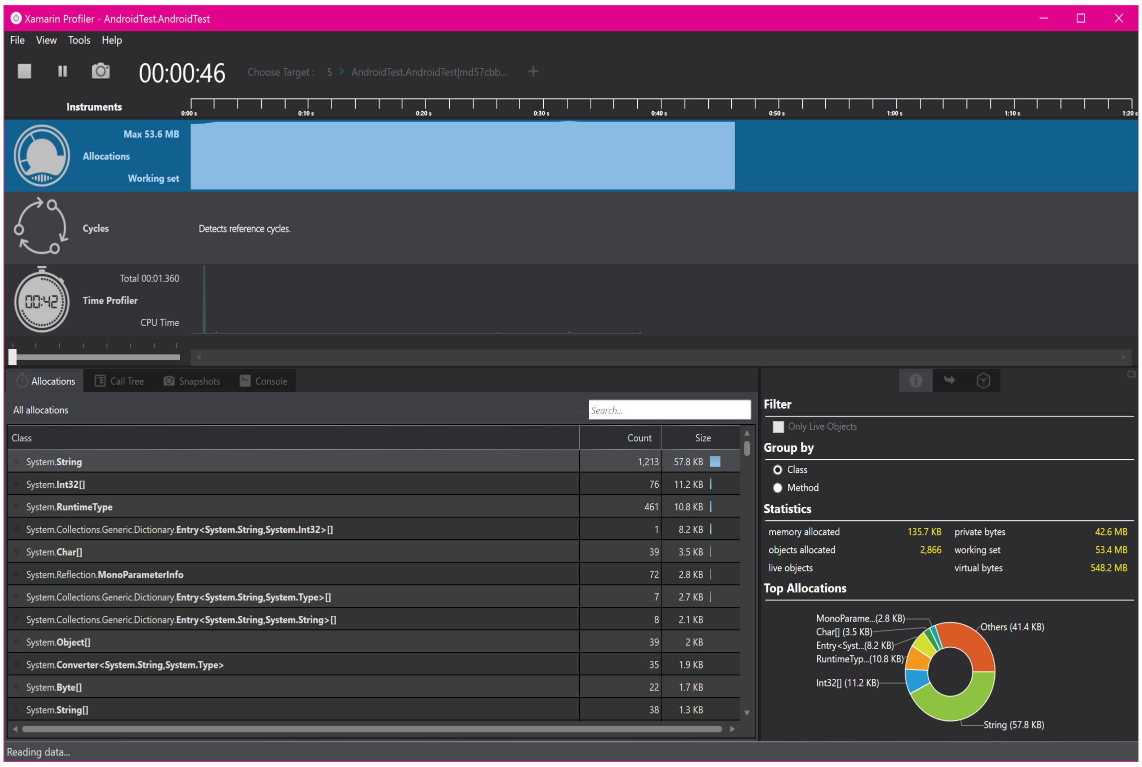

Xamarin Profiler
File View Tools Session Help
00:00:00
Choose target pixel2_pie_9_0_-_api_28 com.companyname.xamari[...]da905cc0a.MainActivity
Instruments
Xamarin Profiler
Choose target
pixel2_pie_9_0_-_api_28 com.companyname.xamari[...]da905cc0a.MainActivity
Available instruments
All Instruments
Includes all available instruments.
Allocations
Analyzes the memory usage by tracking object allocations.
Time Profiler
Profiles a running program on all cores at regular intervals.
Cancel
Previous
Next

fig 6. The actual Xamarin Profiler when started as standalone program



Class	Count	Size
System.**String**	1,213	57.8 KB
System.**Int32[]**	76	11.2 KB
System.**RuntimeType**	461	10.8 KB
System.Collections.Generic.Dictionary.**Entry<System.String,System.Int32>[]**	1	8.2 KB
System.**Char[]**	39	3.5 KB
System.Reflection.**MonoParameterInfo**	72	2.8 KB
System.Collections.Generic.Dictionary.**Entry<System.String,System.Type>[]**	7	2.7 KB
System.Collections.Generic.Dictionary.**Entry<System.String,System.String>[]**	8	2.1 KB
System.**Object[]**	39	2 KB
System.**Converter<System.String,System.Type>**	35	1.9 KB
System.**Byte[]**	22	1.7 KB
System.**String[]**	38	1.3 KB

Filter

Only Live Objects

Group by

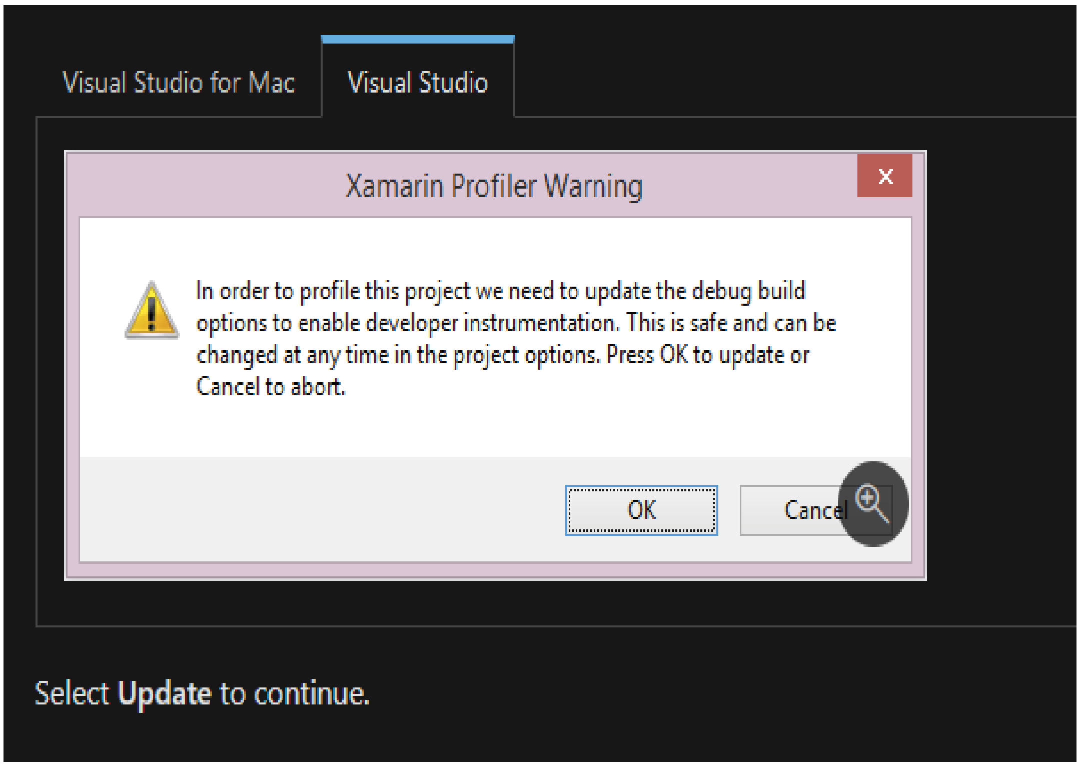
[✓ Class] Method

Statistics

memory allocated	135.7 KB	private bytes	42.6 MB
objects allocated	2,866	working set	53.4 MB
live objects		virtual bytes	548.2 MB

Top Allocations

fig 7. The Xamarin Profiler according to the Xamarin Documentation

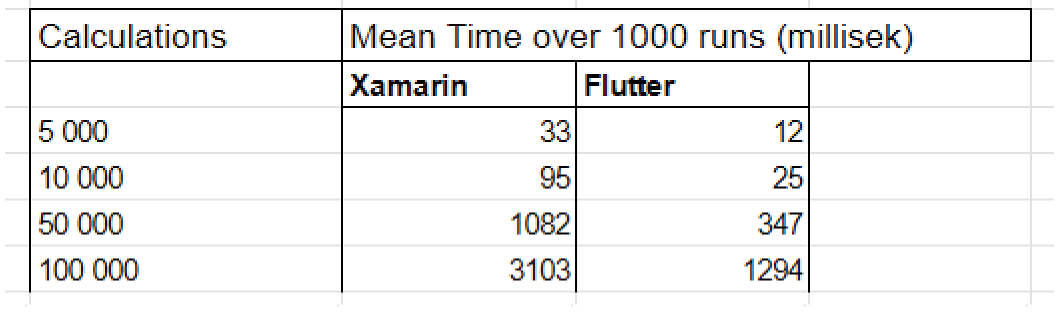


In order to profile this project we need to update the debug build options to enable developer instrumentation. This is safe and can be changed at any time in the project options. Press OK to update or Cancel to abort.

Select **Update** to continue.

fig 8. Misgiving instructions in Xamarin Documentation

Table 1. Mean time for X calculations over 1000 test runs



Calculations	Mean Time over 1000 runs (millisek)	
	Xamarin	**Flutter**
5 000	33	12
10 000	95	25
50 000	1082	347
100 000	3103	1294

Table 3. Computer specifications

Operating System	Windows 10
Service Pack	0
Size	32-bit
Version	10.0.18362
Processor	Intel(R) Core(TM) i5-9600KF CPU @ 3.70GHz
Manufacturer	Intel
Speed	4.6 GHz
Number of Cores	6
CPU ID	BFEBFBFF000906EC
Family	06
Model	9E
Stepping	C
RAM	16 GB
Video Card	NVIDIA GeForce RTX 2060
Manufacturer	Nvidia
Chipset	GeForce RTX 2060
Dedicated Memory	6.0 GB
Total Memory	14 GB
Pixel Shader Version	5.1
Vertex Shader Version	5.1
Hardware T&L	Yes
Vendor ID	10DE
Device	1F08
Plug and Play ID	VEN_10DE&DEV_1F80&SUBSYS_86B01043&REV_A1
Driver Version	27.21.14.5671

Code snippets from applications:

Xamarin CPU test

```
void OnButtonClickedfirst(object sender, EventArgs args)
{
    RunTestsNumberOTimes(5000);
}

void RunTestsNumberOTimes(int numberOfNumbersToSearchForPrimes)
{
    var numberOfTimesToTest = 100;
    var TimeList = new List<double>();
    for (int i = 0; i < numberOfTimesToTest; i++)
    {
        Console.WriteLine("Count: {}" + i);
        var stopwatch = new Stopwatch();
        stopwatch.Start();
        FindPrimeNumber(numberOfNumbersToSearchForPrimes);
        stopwatch.Stop();
        var timeSpan = stopwatch.Elapsed;
        TimeList.Add(timeSpan.TotalMilliseconds);
    }

    double totalTime = 0.0;
    double longestTime = double.NegativeInfinity;
    double shortestTime = double.PositiveInfinity;
    foreach (var time in TimeList)
    {
        totalTime += time;
        if (time < shortestTime)
            shortestTime = time;
        if (time > longestTime)
            longestTime = time;
    }
    var meanTime = totalTime / (double)numberOfTimesToTest;
    Console.WriteLine("TestTimes Avg: {0} Longest: {1} shortest: {2}", meanTime, longestTime, shortestTime);
}
public long FindPrimeNumber(int n)
{
    int count = 0;
    long a = 2;
    while (count < n)
    {
        long b = 2;
        int prime = 1;// to check if found a prime
        while (b * b <= a)
        {
            if (a % b == 0)
            {
                prime = 0;
                break;
            }
            b++;
        }
        if (prime > 0)
        {
            count++;
        }
        a++;
    }

    return (--a);
}
```

Flutter CPU test

```
class _CpuTestingState extends State<CpuTesting> {
  int _foundPrime = 0;

  void startTest(int n) {
    var stopwatch = new Stopwatch();
    var timelist = [];
    int count = 0;
    _foundPrime = 0;
    for (var i = 0; i < 1000; i++) {
      stopwatch.start();
      while (count < n) {
        bool isprime = testPrime(count);
        if (isprime) {
          setState(() {
            _foundPrime++;
          });
        }
        count++;

      }
      stopwatch.stop();
      timelist.add(stopwatch.elapsed.inMilliseconds);
    }
    var totalTime = 0.0;
    for (var time in timelist)
    {
      totalTime += time;
    }
    var meanTime = totalTime / 1000;
    print(meanTime);
  }

  bool testPrime(int number) {
    if (number < 2) {
      return false;
    }
    for (int i = 2; i < number; i++) {
      if (number % i == 0) {
        return false;
      }
    }
    return true;
  }
```