Criteria Based Evaluation of Cross-Platform Development Frameworks

Ali El Tom and Cristian Bogdan
KTH Royal Institute of Technology
Stockholm, Sweden
{eltom,cristi}@kth.se

Tim A. Majchrzak
University of Agder,
Kristiansand, Norway
timam@uia.no

Tor-Morten Grønli
Kristiania University College,
Mobile Techology Lab
Dep. Technology, Oslo, Norway
tor-morten.gronli@kristiania.no

Abstract

Cross-platform development frameworks continue to play an important role in developing of mobile applications. There are a plethora of possible frameworks to choose between, and all claim to give developers an advantage in one or more aspects of the development cycle. To further shed light on this area, our research investigates the selection of mobile cross-platform frameworks by investigating criteria and methods for choosing the best suitable framework for the development context. Our study confirm previous research findings and with that anchor the categorisation for a cross-platform comparison taxonomy. Our research further contribute with a criteria check-list to apply for cross-platform framework selection and showcase its use through a comparative study of a cross-platform application implemented in two frameworks. Although showcasing novelty in the criteria checklist, further research is needed to make the checklist robust and streamlined for public use.

Keywords: Cross-platform, cross-platform development, app, React Native, Xamarin

1. Introduction

Developing mobile applications is complicated by the existence of two major platforms (Android and iOS) and multiple other platforms that need to be supported. There are two major approaches: developing several *native applications* (Biørn-Hansen et al., 2020), one for each platform, and developing using a *cross-platform framework* (CPF). Two thirds of the applications are developed using native application development and one third are developed using Cross Platform Frameworks (CPFs) (Statista, 2022). Using CPFs has been since their inception preferred when a software needs to be developed for several platforms, when the time to market and reducing costs are of high importance (Biørn-Hansen et al., 2018; Raj & Tolety, 2012). Performance and look and feel of mobile applications developed through the leading CPFs has evolved to be close to native, making these approaches attractive to developers and companies (Biørn-Hansen et al., 2020; Fredrikson, 2018)

Developers who decide to opt for CPFs have the problem of choosing from several available CPFs. Helping developers choose the most suitable CFP for their situation is the aim of this paper and many of its related works. For illustration, this paper will use a comparison between the currently popular CPFs *React Native* and *Xamarin*, through an example developed using both CPFs. We employ and further develop an evaluation framework with extensive criteria catalogue.

Our interest in comparing React Native and Xamarin stems from the fact that they are leading CPFs within the respective programming languages, which are also among the most popular. React Native apps are written in JavaScript, used by 67% of developers according to StackOverflow ("Developer Survey Results 20", 2019). The two other most popular CPFs (Cordova and Ioniq) also use JavaScript but Xamarin apps are written in C# (which 31.9% of programmers prefer ("Developer Survey Results 20", 2019)). Both JavaScript and C# (.Net) have large and active developer communities, which makes them popular (Statista, 2022). At GitHub the figures for contributors, used by repositories and GitHub stars are stated in the comparison table between React Native and Xamarin (Table 1).

Table 1. Data from React Native and Xamarin. Forms repository pages on GitHub

	React Native	Xamarin.Forms
GitHub Contributors	2,187	288
Used by Repositories	423,620	67,063
GitHub Stars	88,211	4,580

URI: https://hdl.handle.net/10125/103475
978-0-9981331-6-4

Another perspective in our evaluation was the compatibility and integration of the CPF with modern architectures, namely Redux, developed originally for React JS and other JavaScript frameworks ("Redux - A Predictable State Container for JS Apps", 2020). Redux improves the scalability of classical architectures such as MVC, MVP and MVVM, and is available for C# as Redux.Net (Salles, 2016)

It is challenging to choose among the many available CPFs as they continuously evolve while the ecosystems they need to support evolve at the same time. Both companies and researchers are interested to know how to easier choose between CPFs. Therefore the research questions for this paper are:

RQ1. What are useful criteria when choosing a cross-platform framework?

RQ2. Which methods can help a company or developer choose cross-platform frameworks?

The remainder of the paper is structured as follows. In Section 2, we draw the background of our work. We then present evaluation methods for cross-platform development frameworks in Section 3. Previous studies are summarizes in Section 4. Our results are given in Section 5. We discuss our findings in Section 6 before drawing a conclusion in Section 7.

2. Background

2.1. Performance Evaluations of React Native

Biørn-Hansen et al. (2020) evaluated Native Android, React Native, Flutter, Ionic, NativeScript and MAML/MD2. Their performance metrics where time-to-completion (TTC), memory consumption (PreRAM and ComputedRAM) and CPU usage. They studied the features Accelerometer, Contacts, File System Access and Geolocation. They conclude cross-platform development may lead to decreased performance in comparison with native, but at the same time some CPFs may perform equally or even better than native, depending on feature or metric. When it comes to React Native, it performed better than native with regards to the overall weight for ComputedRam, but less in TTC, CPU and PreRAM.

Rieger and Majchrzak (2019) compared native to Web apps and the CPFs PWA, PhoneGap and React Native. They found that React Native has 34 APIs with access to platform functionalities and a low performance overhead, as React Native delegates heavy performance computations to the native environment. Fredrikson (2018) made a comparison between Progressive Web Applications (PWA), React Native and native Android app and identified 42 features that were available in native Android, all of which were available in React Native except one rarely used, while only 35 features where available PWA.

2.2. Performance Evaluations of Xamarin

Delia et al. (2017) made a performance analysis and assessed the processing speed of Web apps and several CPFs (Xamarin, Cordova, Titanium, NativeScript and Corona) with native (Android and iOS). They have also analysed and compared the iOS and Android application separately. The native iOS app was found to be more efficient than the native Android app. The iOS app developed with Xamarin showed also better results that the Android app developed with Xamarin. The performance of the android app developed in Xamarin was close to and even better than the Android app developed in native Java, while in iOS the native performance was much better than Xamarin. The frameworks analysed showed better results than Xamarin in Android, while in iOS the results of Xamarin was better than the interpreted frameworks assessed.

Corbalan et al. (2018) compared native Android with several interpreted and cross-compiling CPFs including Xamarin. They studied the impact on CPFs and energy consumption, where the Xamarin applications had close to native (Android SDK) energy consumption for all three applications (video playback, audio playback and intensive processing applications.

Ebone et al. (2018) compared native Android and iOS with several CPFs, among them Xamarin. Their findings is that apps developed with Xamarin can look and behave exactly like native apps with a performance close to native. Their observations included the metrics building time, rendering time, Uwe response time, memory usage and app size. When observing building time they noticed that both native (iOS and Android) and apps developed with Xamarin have a building time that grows proportionally with respect to the size of the view, saying that they have a similar behaviour constructing the view hierarchy, which represents the whole view before a portion of the view is rendered. The native Android app has near constant rendering time with respect to the view size, which was similar with the application built with Xamarin.Android. Android and iOS applications built with Xamarin.Forms were much slower for the large views. Xamarin.Forms was significantly slower in total Uwe response time than native Android and iOS apps for larger views. The observation on memory usage showed a similar behaviour between Xamarin and native in that memory usage increases in proportion to the Uwe size. And lastly that app size of the Android and iOS apps developed

with Xamarin were larger than native Android and iOS.

Yahya-Imam et al. (2019) in a literature study evaluated several CPFs: Codename One, Xamarin, Titanium, PhoneGap and Sensa Touch. Regarding Xamarin their performance finding is that Xamarin delivers native performance but load times are slightly slower. Xamarin has close to native performance and a native look and feel as confirmed by several articles (Corbalan et al., 2018; Delia et al., 2017; Ebone et al., 2018; Yahya-Imam et al., 2019).

3. Evaluation Methods For Choosing Cross-Platform Frameworks

3.1. The Choice of the Evaluation Framework

According to Biørn-Hansen et al. (2018), most researchers are evaluating cross-platform frameworks (CPF) based on *state of the art* frameworks from older research, which are not valid any more, while developers are communicating and showcasing their industrial comparisons frequently on the latest frameworks through blogs, websites and conferences. They recommend actively using non-academic contributions and regard that as high importance for understanding and properly evaluating and comparing the true and latest state of the art technologies. We are following their advice in this paper.

Rieger and Majchrzak (2019), define an assessment framework for evaluating CPFs for different situations and with an extensive criteria catalogue. This builds upon methods developed first by Heitkötter et al. (2012) in 2012 and the research paper in 2016 by Rieger and Majchrzak (2016). They tested their assessment framework by evaluating and comparing native apps, Web Apps and the CPFs PhoneGap, PWA and React Native. The framework proposes comprehensive weight based evaluation framework with an extensive criteria catalogue of 33 criteria that can help in choosing cross-platform development approaches not only for smart phones but also for other app enabled devices, such as smart watches and Smart TVs. Example weight profiles are provided, which can be updated by companies and individuals based on their specification and needs. The 33 evaluation criteria are divided in four perspectives or groups which are Infrastructure (Table 2), Development (Table 3), App (Table 4) and User Perspective (Table 5); all tables follow the work by Rieger and Majchrzak (2019).

Each criteria receive a score of 0 to 5. There is a weight of 100 to be distributed between the 33 criteria, with each value between 0 and 7. The score of a cross-platform framework is then calculated based

Table 2. Infrastructure perspective

Criterion	
I1	License
I2	Target platforms
I3	Development platforms
I4	Distribution channels
I5	Monetisation
I6	Internationalisation
I7	Development platforms

Table 3. Development perspective

Criterion	
D1	Development Environment
D2	Preparation Time
D3	Scalability
D4	Development Process Fit
D5	UI Design
D6	Testing
D7	Continuous Delivery
D8	Configuration Management
D9	Maintainability
D10	Extensibility
D11	Custom Code Integration
D12	Pace of Development

Table 4. App perspective

Criterion	
A1	Hardware Access
A2	Platform Functionality
A3	Connected Devices
A4	Input Heterogeneity
A5	App Life Cycle
A6	System Integration
A7	Security
A8	Robustness

Table 5. Usage perspective

Criterion	
U1	Look and Feel
U2	Performance
U3	Usage patterns

Equation 1, with S (score), i_j (infrastructure criteria score), d_j (development criteria score), a_j (app criteria score) and u_j (user perspective criteria score), $w_{i,j}$, $w_{d,j}$, $w_{a,j}$ and $w_{u,j}$ being the weights for each criteria.

$$S = \frac{\sum_{j=1}^{7} * w_{i,j} * i_j + \sum_{j=1}^{12} * w_{d,j} * d_j}{100} + \frac{\sum_{j=1}^{10} * w_{a,j} * a_j + \sum_{j=1}^{4} * w_{u,j} * u_j}{100} \quad (1)$$
$$\sum_k w_k = 100$$

The authors also recommend that the final score is rounded to two decimal digits. The higher and closer the score is to 5 the better is the cross-development framework.

3.2. Framing a Use Case

Hudli et al. (2015), Lachgar and Abdali (2017), Latif et al. (2016), and Raj and Tolety (2012) used few criteria and did not have a well described evaluation method. The (Rieger & Majchrzak, 2019) evaluation framework covers the criteria used in previous scientific and industrial research that we reviewed. It has also detailed description not only from the above mentioned paper but from previous papers by the same authors, which makes it easy to understand and implement. What is lacking, though, is a per-criteria checklist that helps in the evaluation and reaching the score of each criterion of each framework. We propose such a per-criteria checklist in this paper.

As the basis for the weights preference in the evaluation framework, we use a specific project requirements presented by a company (Decerno AB, Sweden) as a use case. These are the need for web code reuse, high performance, application state implementation using a Redux store, local storage in the web browser and license. An application called *DeTodo* has been developed and evaluated.

In addition, we developed a React Native and Xamarin apps with application state implementation using a Redux store, and evaluate the Redux store implementation in both CPFs.

The criteria used for the evaluation of the implementation of Redux store are the community, maintainers, architecture and design guidelines, learning materials, persistence tools, debugger tools, code kit for minimising boilerplate code and lastly lines of code.

4. Previous Comparisons of React Native and Xamarin

Biørn-Hansen, Grønli, Ghinea, and Alouneh (2019), conducted an empirical study of Cross-Platform mobile development in industry, surveying 101 developers, and having many CPFs (including React Native and Xamarin). The purpose was to report what the industry thinks of the CPFs, with metrics that were familiarity, interest of exploring, hobby usage, professional usage and issues. In their survey React Native scored the highest with regards to the participants interest in learning (4.28 out of 5), which matches the industry statistics provided by StackOverflow ("Developer Survey Results 20", 2019). Xamarin.Forms received a result of only 1.64 out of 5 in the participants interest, which was lower than other popular frameworks like PhoneGap/Cordova, NativeScript and Ionic.

Biørn-Hansen, Grønli, and Ghinea (2019) also studied animations in CPFs. They developed 8 apps each in native (Android and iOS), hybrid (Ionic), interpreted (React Native) and cross-compiled (Xamarin) approaches. Their focus was on performance testing of graphical user interfaces. The metrics they studied where FPS, CPU usage, Device memory usage (RAM) and GPU memory usage. React Native and Xamarin Forms differed very little in FPS and CPU overall in animation, navigation transition and side menu performance in Android. React Native had much better results in Memory consumption and GPU in Android. In iOS React Native also had better results than Xamarin in regards to Memory usage overall.

An important takeaway from Biørn-Hansen, Grønli, and Ghinea (2019) is that if a specific framework is better performing in Android it could be the opposite in iOS. Therefore. more specific product requirements may help decide which framework to choose. It is also challenging to choose a framework based on performance testing animation since both native and CPFs behave differently in iOS and Android. For their tests they point out that React Native was the most performant CPF in regards to Android. In iOS it was difficult to point out a winner, React Native was better overall in memory usage, but had a much higher CPU usage in animations (30% more than Xamarin), while there was very slight non significant differences otherwise. In future research it would be of interest to study the navigation transition performance of JavaScript-based frameworks such as React Native in relation to device resource usage and user experience. Also of interest is more research that correlates between performance and user perception.

Avdic (2019) compared React Native and Xamarin, as the Tetra Pak company wanted to choose one of them for developing their internal employees mobile application. The author focused on maintainability of the system and performance, but the main reason the company chose Xamarin in the end was a maintainability reason, due to the company's larger knowledge base and more experience with Xamarin, ASP.Net and C#.

Satei (2019) compared React Native, Xamarin and Flutter with regards to video-oriented mobile applications such as Netflix, Amazon Prime, YouTube and HBO. Since it is not possible to integrate an existing native app with Xamarin, the subscription costs related to using Xamarin and no advantages witnessed over React Native and Flutter, Xamarin was dropped early from the evaluation. However, Vishal and Kushwaha (2018) conclude that Xamarin.Forms is more prominent than React Native in development environment and pre-existing components. Also with regards to compilation similar to those by Agarwal

Table 6. React Native Vs Xamarin, Google search 9 July 2020

Result of comparison	Articles	Reference
Chose React Native as the best of the two	2	Sinha, 2020; Your-Team-India, 2020
Chose Xamarin as the best of the two	0	-
Equally good, choice of CPF depends on specific needs and project requirements	4	Agarwal, 2020; Kanti, 2020; Rees, 2020; Vignesh, 2020
Equally good, choice of CPF depends on programming language the developer team has more experience in or more comfortable with	4	Barnes, 2020; Khan, 2020; Machanda, 2020; Sharma, 2020

(2020), Avdic (2019), and Machanda (2020), Vishal and Kushwaha (2018) mention Xamarin outperforming React Native in compilation methods.

To get non-academic insights, a search was made in Google for "React Native vs Xamarin" within the time period 1 January 2020 to 9 July 2020. After analysing the top ten pages, the following results in Table 6 were noted. 8 out of 10 of these industry articles state that the choice of CPF could depend on the programming language the team has more experience with.

5. Results

5.1. Criteria Selection

In Table 7 we present the mapping of the requirements with the criteria. In the criteria catalogues Infrastructure perspective, the criterion I2 (Target platforms) the platforms that are important to cover are Android, iOS and Web. In the Development perspectives criteria D3 (Scalability), D6 (Testing) and D12 (Pace of Development), Redux has been added as checklist item in the per-criteria evaluation form. Web code reuse has been evaluated in the checklist items of the criteria D9 (Maintainability). Performance and User experience are covered in the checklist items of the Usage Perspective criteria U1 (Look and Feel) and U2 (Performance). License and Costs are covered in the checklist items of the Infrastructure Perspective criteria I1 (License) and I2 (Development Environment). Local Storage is evaluated in the App Perspective criteria A2 (Platform Functionality).

These following criteria from Rieger and Majchrzak (2019) are not relevant for the use case and also because of time limitation they were not evaluated and got zero weight: D8 (Configuration Management), A3 (Connected Devices), A4 (Input Heterogeneity), A5 (Output Heterogeneity) and A10 (Degree of Mobility).

Table 7. Requirements mapping to criteria

Requirement	Criteria
App platforms iOS, Android and Web	I2
Web code reuse	D9
Redux state management	D3, D6 and D12
Performance and User Experience	U1 and U2
Local Storage	A2
License and Costs	I1 and I2

In our proposed per-criteria checklist form, each checklist item is a presumption that gets scored from 0 (zero) to 5, where 0 (zero) means *Not relevant*. 1 is *Strongly Disagree*, 2 is *Disagree*, 3 is *Neutral*, 4 is *Agree* and 5 is *Strongly Agree*. For example one checklist item in I2 is: "Using the CPF developers can also develop Web apps with ease, there is good documentation on how to do that and others in the industry have used it". If it is not possible then the developer will give it 1, otherwise score it accordingly. Then for each criteria the checklist items scores are calculated as the mean score of the total checklist items in the criteria and then added to the evaluation framework by Rieger and Majchrzak (2019). The per-criteria checklist items cover most the things that Rieger and Majchrzak (2019) mention as "must" or "should" that are found or that are important for the use case. Some presumptions were also added based on the experience developed and based on the use case. In each criteria checklist item table, a column was added for filling the source of reference. The per-criteria checklist is comprehensive since it covers all 33 criteria by Rieger and Majchrzak (2019), and we believe that such a time-consuming qualitative evaluation exercise will best be made by a team of developers.

5.2. Per-Criteria Checklist Evaluation Results

In Tables 8 (Infrastructure), 9 (Development), 10 (App) and 11 (Usage) we present the mean results of the proposed per-criteria checklist evaluation forms for both React Native (referred to in the table head as *R*) and Xamarin (referred to in the table head as *X*).

Table 8. Infrastructure Perspective per-criteria checklist means results

	R	X
I1 License (Mean score)	5.0	3.0
I2 - Target Platforms (Mean score)	4.8	4.0
I3 - Development Platforms (Mean score)	4.3	3.3
I4 - Distribution Channels (Mean score)	4.8	4.6
I5- Monetisation	4.7	4.3
I6- Internationalisation (Mean score)	4.7	5.0
I7 - Long-term feasibility (Mean score)	4.6	3.2

Table 9. Development Perspective per-criteria checklist means results

	R	X
D1 - Development Environment (Mean score)	5.0	4.5
D2 - Preparation Time (Mean score)	4.8	4.0
D3 - Scalabilty (Mean score)	5.0	3.3
D4 - Development Process Fit (Mean score)	4.8	3.5
D5 - UI Design (Mean score)	4.3	4.7
D6 - Testing (Mean score)	4.8	4.2
D7 - Continuous Delivery (Mean score)	3.3	4.0
D8 - Configuration Management3 (Mean score)	0.0	0.0
D9 - Maintainability (Mean score)	4.6	3.4
D10 - Extensibility (Mean score)	5.0	3.0
D11 - Custom Code Integration (Mean score)	4.0	3.0
D12 - Pace of Development (Mean score)	4.6	3.4

Table 10. App Perspective per-criteria checklist means results

	R	X
A1 - Hardware Access (Mean score)	4.9	4.5
A2 - Platform Functionality (Mean score)	5.0	4.9
A3 - Connected Devices (Mean score)	0	0
A4 - Input Heterogeneity (Mean score)	0	0
A5 - Output Heterogeneity (Mean score)	0	0
A6 - App Life Cycle (Mean score)	3.7	5.0
A7 - System Integration (Mean score)	4.5	4.3
A8 - Security (Mean score)	4.4	4.8
A9 - Robustness (Mean score)	4.3	4.3
A10 - Degree of Mobility (Mean score)	0	0

Table 11. Usage Perspective per-criteria checklist means results

	R	X
UI - Look and Feel (Mean score)	3.9	4.0
U2 - Performance (Mean score)	4.0	3.8
U3 - Usage Patterns (Mean score)	3.9	4.0
U4 - User Authentication (Mean score)	3.5	3.3

In the proposed per-criteria checklist evaluation form each criteria has a set of checklist items that are scored from 0 to 5. The total score for each criteria is calculated as the mean score of the items in the criteria.

The comparison results are presented in Table 12. The scores are calculated based on the results from the per-criteria checklist evaluation forms.

5.3. Redux Implementation Comparison

It was possible to use Redux in React Native projects similar to (non-native) React JS. Compared to the Redux library available for JavaScript, the Redux.NET library has extremely low download numbers and an almost non existing community. Very few developers have expressed the interest in using the architecture in Xamarin One of the main reasons according to the senior .NET developer Christer van der Meeren is the very large boilerplate code needed with C# and

Table 12. Criteria catalogue and evaluation results

Criteria	Weight	R	X
Infrastructure			
I1 - License	5	5	3
I2- Target Platforms	6	5	4
I3- Development Platforms	2	4	3
I4- Distribution Channels	2	5	5
I5- Monetisation	1	5	4
I6- Internationalisation	1	5	5
I7 - Long-term feasibility	5	5	3
Development			
D1 - Development Environment	7	5	5
D2 - Preparation Time	7	5	4
D3 - Scalabilty	7	5	3
D4 - Development Process Fit	2	5	4
D5 - UI Design	4	4	5
D6 - Testing	3	5	4
D7 - Continous Delivery	3	3	4
D8 - Configuration Management	0	0	0
D9 - Maintainability	4	5	3
D10 - Extensibility	2	5	3
D11 - Custom Code Integration	2	4	3
D12 - Pace of Development	4	5	3
App			
A1 - Hardware Access	4	5	5
A2 - Platform Functionality	5	5	5
A3 - Connected Devices	0	0	0
A4 - Input Heterogeneity	0	0	0
A5 - Output Heterogeneity	0	0	0
A6 - App Life Cycle	2	4	5
A7 - System Integration	3	5	4
A8 - Security	3	4	5
A9 - Robustness	2	4	4
A10 - Degree of Mobility	0	0	0
Usage			
U1 - Look and Feel	5	4	4
U2 - Performance	6	4	4
U3 - Usage Patterns	2	4	4
U4 - User Authentication	1	4	3
Total	**100**	**4.65**	**3.96**

Xamarin ("What's the status of this project?", 2017). Table 13 presents the comparison results based on the experience of implementing Redux on React Native and Xamarin.Forms.

5.4. Reuse Of Web Code

The DeToDo app developed in the paper used react-native-web and thereby delivered a React Native web app that can be deployed to a website. If one starts with a React JS web app, there have been reports of reusing the between 60% (Spooman007, 2019) and 90% (Munro, 2018) of the web code.

Such reuse is not possible in Xamarin as it uses another programming language (C#). It is possible to use WebView (Microsoft, 2020b) to display web content in the mobile application

Table 13. React Native and Xamarin – Redux comparison

Criteria	R	X
1. Large and active community	Yes	Not found
2. Active and supportive NPM/Nuget library maintainers	Yes	No
3. Clear architecture and design guidelines	Yes	No
4. Proper detailed official documentation and tutorials	Yes	No
5. Redux persistance tool/package that is available and easy to integrate	Yes	No
6. Redux Debugger tools for time-machine debugging and testing	Yes	Yes, not used due to not being updated since 2015
7. Redux code kit - to simplify and reduce boiler plate code	Yes	No
8. Total Redux Lines of Code	∼60	∼140

5.5. Local Device Storage

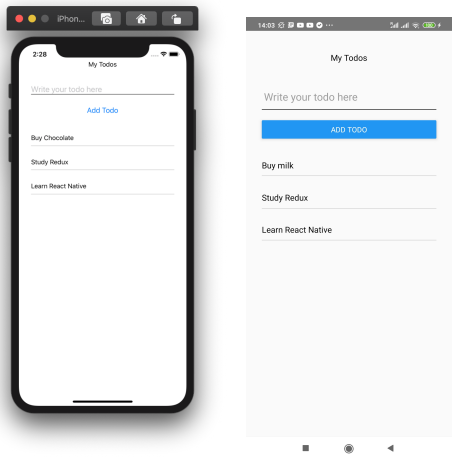
DeToDo uses React Native Async Storage ("GitHub: react-native-async-storage / async-storage", 2022), which is the equivalent of Local storage from the Web. It was very simple to use together with Redux through the popular community middleware package redux-persist ("Redux Persist", 2020) and the official Redux Toolkit (Redux, 2020).

Xamarin.Forms supports the possibility for persisting data to local device storage through file handling by storing and accessing files using the .NET Standard library or embedded resources, or using the local database engine SQLite (Microsoft, n.d.). The code written in C# for Xamarin is then compiled to native byte code on each platform that accesses the native file APwe of that platform (Biørn-Hansen et al., 2018). The System.IO classes of the .NET Standard library are used to access the files system of each platform (Microsoft, 2020a)

5.6. License

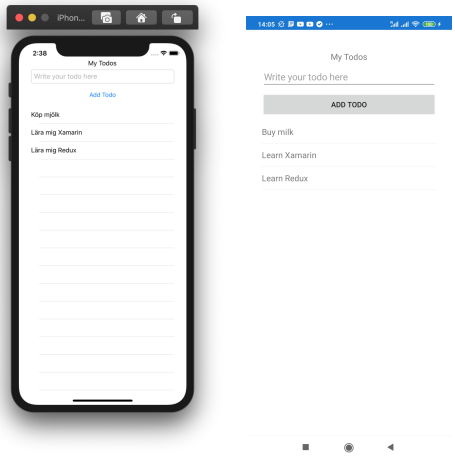
React Native is free to use and is open source. It is distributed under a permissive MIT Licence whereby permission is granted to use it free of charge without restrictions ("GitHub: react-native/LICENSE", 2022). React Native apps can be developed using any IDE, several of these are free. The most popular IDE for mobile developers is Visual Studio Code, and it is free ("Developer Survey Results 20", 2019).

Xamarin SDK is free (".NET is Free", 2022) and open-source under the permissive MIT license ("GitHub: Xamarin.Forms/LICENSE", 2022). It is delivered part of Microsoft Visual Studio, which is also the most common way to develop apps with Xamarin (Yahya-Imam et al., 2019). Visual Studio could be used free of charge with the Visual Studio 2019 Community version. But that can be used for initial learning only and not for business since one can not profit out of the apps developed and it is very limited to use for companies.



(a) React Native iOS (b) React Native Android

Figure 1. DeToDo app developed with React Native



(a) Xamarin.Forms iOS (b) Xamarin.Forms Android

Figure 2. DeToDo app developed with Xamarin

5.7. Performance Comparison: APK Download Size

In the DeToDo applications that were developed due to time limitations the only metric that was noted and compared is the final application download size. As mentioned recently in 2020 by Biørn-Hansen et al.

(2020) this metric is one of the most important when taking technical decisions. In a more comprehensive performance study, one would need a more detailed project specification and requirements of the type of performance evaluation that needs to be made, and since the data collection should be made manually. With regards to APK download size, React Native delivered a release APK file download size of 6.7 MB which was lighter than Xamarin.Forms, which was 8.2 MB.

6. Discussion

Before evaluating any CPF we learned that one needs to look deeper at the project requirements. One needs to understand whether the non-functional requirements provide preferences for a certain programming language. It is also of interest whether the new project builds upon another one that is built with a certain programming language. This would be the first step that could lead to a fast decision into what CPF to choose. The use case presented by our partner company was of a software development team of web front-end developers that uses today mostly a JavaScript framework in their projects, are interested in a fitting license and using local storage while having optimum app performance. Most important they want to save time and resources by reusing in the mobile apps most of their code used in their web projects including the business logic and architecture made in Redux.

Based on the clear programming language preference of the company, React Native would be choose initially, especially since one of their important requirements was code-reuse from other JavaScript based web projects.

The findings after developing the mobile applications with React Native and Xamarin are that in React Native, developers can reuse more code developed in web-projects with JavaScript than Xamarin. In regards to the application state implementation using Redux store, it was experienced to have equally similar code in JavaScript (React Framework) and React Native, so there would be up to 100% code reuse for that. While the implementation of Redux store in Xamarin is different than in JavaScript due to the different programming languages one can not reuse such code between Web and Xamarin.

State management principles such as the today popular Redux can be implemented in different programming languages. However, it is harder to implement Redux in C# compared to JavaScript due to the lack of active developer usage and support.

When it comes to projects that do not have programming language preferences, we developed further the evaluation framework by Rieger and Majchrzak (2019). It is important to note that not all criteria they presented are important for all types of projects; therefore, we found it essential to omit theose that are not applicable. In this paper and for the above mentioned use case we configured the weights of the evaluation framework.

Our contribution is a per-criteria evaluation form in order to be able to evaluate any CPF. For us, it made it much easier and systematic to make the comparison, especially since the evaluation framework (Rieger & Majchrzak, 2019) was truly comprehensive but lacked such checklist. As they had suggested the per-criteria checklist is needed for better operationalisation of own assessments.

It was hard to make the per-criteria form generic due to limited time, so we made it fit the general case of web developers who want to develop native mobile applications using a CPF. We believe the per-criteria evaluation form can be configured for other use cases, but that would left for future research. The per-criteria evaluation form was used to evaluate React Native and Xamarin separately based on scientific research and latest online references, in addition to the applications developed. Subsequently, the scores calculated in the evaluation forms were used in the weight based evaluation framework by Rieger and Majchrzak (2019). For that specific use case React Native was the better choice with a score of 4.65 while Xamarin.Forms received a score of 3.96.

The approach of using the developed per-criteria evaluation form and the weight based evaluation framework is very time-consuming because the checklist, which covers the 33 criteria, has at this point in time up to 162 checklist items that needs to be evaluate. We recommend that the evaluation is performed by a team of developers with enough time to complete all items thoroughly.

The development of checklists, the validity of checklist items, and the validity of the checklist-based evaluation are current limitations. While the criteria catalogue has been developed in a scientifically rigorous manner, the same level of rigour is hard to achieve for checklists due to their practical nature and the volatility of the underlying technologies and practices. Working with checklists and evaluating has a subjective component due to the qualitative way in which they need to be done, even if performed in teams.

A reasonable way to address these limitations is to aim for scale and for consensus. With a sufficiently large community, evaluations would be hardened and generalizable advice would become applicable to *most* settings, organizations, and projects.

7. Conclusion

The goal of this paper was to identify the criteria and methods that can be used to evaluate which cross-platform framework (CPF) to choose from for building a mobile application to Android and iOS.

Answering the first research question *RQ1. What are useful criteria when choosing a cross-platform framework?*, the most wide scale criteria catalogue used for evaluating CPFs for mobile application development was identified during the literature study. It has been developed by Rieger and Majchrzak (2019). This criteria catalogue has 33 criteria divided in 4 perspectives Infrastructure, Development, App and Usage . For our partner company all criteria were useful to evaluate except D8 (Configuration Management), A3 (Connected Services), A4 (Input Heterogeneity), A5 (Output Heterogeneity) and A10 (Degree of Mobility) as explained in the Results section.

Regarding the second research question: *RQ2. Which methods can help a company or developer choose cross-platform frameworks?*, the most comprehensive method for comparing CPFs identified was the weight based evaluation framework by Rieger and Majchrzak (2019), which covers the 33 criteria catalogue mentioned above. Rieger and Majchrzak (2019) have developed this weight based evaluation framework through several years scientific research, from 2012 (Heitkötter et al., 2012; Rieger & Majchrzak, 2016). Their evaluation framework is well documented but it is missing a step by step per-criteria checklist that can be used when evaluating the CPFs. We contributed with a per-criteria checklist form as proposal and used it to evaluate two CPFs for a specific use-case.

Evaluating two or more CPFs with both the proposed per-criteria checklist form and the evaluation framework is very time consuming. Therefore, further testing and development of the proposed per-criteria checklist form is left for future work.

References

.NET is Free. (2022). https://dotnet.microsoft.com/platform/free

Agarwal, R. (2020). React native vs xamarin: Which cross-platform framework to choose in 2020? https://www.algoworks.com/blog/react-native-vs-xamarin-2020/

Avdic, D. (2019). *React native vs xamarin-mobile for industry* [Master's thesis, Department of Electrical and Information Technology Faculty of Engineering, LTH, Lund University]. Lund, Sweden.

Barnes, G. (2020). React native vs xamarin: 5 surprising similarities you should know. https://devcount.com/react-native-vs-xamarin/

Biørn-Hansen, A., Grønli, T.-M., & Ghinea, G. (2018). A survey and taxonomy of core concepts and research challenges in cross-platform mobile development. *CSUR*, *51*(5), 1–34.

Biørn-Hansen, A., Grønli, T.-M., & Ghinea, G. (2019). Animations in cross-platform mobile applications: An evaluation of tools, metrics and performance. *Sensors*, *19*(9), 2081.

Biørn-Hansen, A., Grønli, T.-M., Ghinea, G., & Alouneh, S. (2019). An empirical study of cross-platform mobile development in industry. *Wireless Communications and Mobile Computing*, *2019*.

Biørn-Hansen, A., Rieger, C., Grønli, T.-M., Majchrzak, T. A., & Ghinea, G. (2020). An empirical investigation of performance overhead in cross-platform mobile development frameworks. *EMSE*, *25*, 2997–3040.

Corbalan, L., Fernandez, J., Cuitiño, A., Delia, L., Cáseres, G., Thomas, P., & Pesado, P. (2018). Development frameworks for mobile devices: A comparative study about energy consumption. *Proc. 5th MOBILESoft*, 191–201.

Delia, L., Galdamez, N., Corbalan, L., Pesado, P., & Thomas, P. (2017). Approaches to mobile application development: Comparative performance analysis. *2017 Computing conference*, 652–659.

Developer survey results 20. (2019). https://insights.stackoverflow.com/survey/2019%5C#key-result19

Ebone, A., Tan, Y., & Jia, X. (2018). A performance evaluation of cross-platform mobile application development approaches. *IEEE/ACM 5th MOBILESoft*, 92–93.

Fredrikson, R. (2018). *Emulating a native mobile experience with cross-platform applications* [Master's thesis, School of EECS, KTH]. Stockholm, Sweden.

Github: React-native-async-storage / async-storage. (2022). https://github.com/react-native-async-storage/async-storage

Github: React-native/license. (2022). https://github.com/facebook/react-native/blob/main/LICENSE

Github: Xamarin.forms/license. (2022). https://github.com/xamarin/Xamarin.Forms/blob/main/LICENSE

Heitkötter, H., Hanschke, S., & Majchrzak, T. A. (2012). Evaluating cross-platform development approaches for mobile applications. *WEBIST*, 120–138.

Hudli, A., Hudli, S., & Hudli, R. (2015). An evaluation framework for selection of mobile app development platform. *Proc. 3rd Int. Workshop on Mobile Development Lifecycle*, 13–16.

Kanti, R. (2020). React native vs xamarin: Which technology is in the race? https://www.qsstechnosoft.com/xamarin-vs-react-native

Khan, D. (2020). Xamarin vs react native – which is better? https://www.folio3.com/mobile/blog/xamarin-vs-react-native-which-is-better/

Lachgar, M., & Abdali, A. (2017). Decision framework for mobile development methods. *Int J Adv Comput Sci Appl (IJACSA)*, *8*(2).

Latif, M., Lakhrissi, Y., Es-Sbai, N., et al. (2016). Cross platform approach for mobile application development: A survey. *IT4OD*, 1–5.

Machanda, A. (2020). React native vs xamarin: Which is a better mobile framework. https://www.netsolutions.com/insights/react-native-vs-xamarin-answering-the-million-dollar-question/

Microsoft. (n.d.). Xamarin.forms local data storage. https://docs.microsoft.com/en-us/xamarin/xamarin-forms/data-cloud/data/

Microsoft. (2020a). File handling in xamarin.forms. https://docs.microsoft.com/en-us/xamarin/xamarin-forms/data-cloud/data/files?tabs=macos

Microsoft. (2020b). Xamarin.forms webview. https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/webview?tabs=macos

Munro, J. (2018). 90% code reuse in production with react native for web. https://www.datacamp.com/community/tech/porting-practice-to-web-part1

Raj, C. R., & Tolety, S. B. (2012). A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. *INDICON*, 625–629.

Redux. (2020). Redux toolkit - the official, opinionated, batteries-included toolset for efficient redux development. https://redux-toolkit.js.org

Redux - a predictable state container for js apps. (2020). https://redux.js.org

Redux persist. (2020). https://github.com/rt2zz/redux-persist

Rees, J. (2020). Flutter vs react-native vs xamarin. https://dzone.com/articles/flutter-vs-react-native-vs-xamarin

Rieger, C., & Majchrzak, T. A. (2016). Weighted evaluation framework for cross-platform app development approaches. *EuroSymposium*, 18–39.

Rieger, C., & Majchrzak, T. A. (2019). Towards the definitive evaluation framework for cross-platform app development approaches. *JSS*, *153*, 175–199.

Salles, G. (2016). Redux.net. https://github.com/GuillaumeSalles/redux.NET

Satei, M. (2019). *Ott video-oriented mobile applications development using cross-platform ui frameworks* [Master's thesis, School of EECS, KTH]. Stockholm, Sweden.

Sharma, S. (2020). Xamarin vs react native: Pick the right platform in 2020. https://www.appventurez.com/blog/xamarin-vs-react-native/

Sinha, D. (2020). React native vs xamarin: What to choose for cross-platform app development? https://www.techaheadcorp.com/blog/react-native-vs-xamarin/

Spooman007. (2019). Reuse business logic between react and react native. https://dev.to/spoman007/reuse-business-logic-between-react-and-react-native-5gj0

Statista. (2022). Cross-platform mobile frameworks used by developers worldwide 2019. https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/

Vignesh. (2020). Flutter vs. react native vs. xamarin - detailed comparison. https://yourstory.com/mystory/flutter-vs-react-native-vs-xamarin

Vishal, K., & Kushwaha, A. S. (2018). Mobile application development research based on xamarin platform. *4th ICCS*, 115–118.

What's the status of this project? (2017). https://github.com/GuillaumeSalles/redux.NET/issues/69%5C#issuecomment-318020015

Yahya-Imam, M. K., Palaniappan, S., & Ghadiri, S. M. (2019). Investigation of methodical framework for cross-platform mobile application development: Significance of codename one. *IJCAET*, *11*(4-5), 439–448.

Your-Team-India. (2020). React native vs xamarin – which cross-platform is better? https://www.yourteaminindia.com/blog/react-native-vs-xamarin/