Baseline Requirements for Comparative Research on Cross-Platform Mobile Development: A Literature Survey

Andreas Biørn-Hansen¹, Tor-Morten Grønli¹ and Gheorghita Ghinea²

¹ Mobile Technology Lab, Faculty of Technology, Westerdals Oslo ACT, Oslo, Norway
² Department of Computer Science Brunel University, Uxbridge, United Kingdom

Abstract

Technical implementations are common in computing research to objectively assess hypotheses. In mobile computing, and more specifically within research on cross-platform mobile development, such implementations are usually in the form of mobile apps. Due to the lack of a common ground for research on app development, studies tend to lack depth and miss out on possible contributions. In an attempt to better the situation, we propose a technical baseline for future research on cross-platform app development to draw from based on previous studies’ technical implementations. We assess and scrutinize existing literature to find trends, and use the generated knowledge to lay out the baseline proposal.

1 Introduction

Cross-platform mobile development is a popular alternative to the traditional native development approach, according to Viennot et al. [1] and Ali and Mesbah [2]. While cross-platform development tend to differ greatly from native development [3], the common goal is to produce applications for mobile devices. The native approach requires specialized platform knowledge, as each platform (e.g. Android, iOS and Windows Phone) introduce their own separate programming language, design guidelines, programming interfaces, development environments and more [4]. Thus, developing an app for each aforementioned platform would effectively require proficiency in Java, Swift / Objective-C and C# [5], as well as deep understanding of the Apple Human Interface Guidelines, Android’s Material Design and Microsoft’s Modern UI [4].

Due to severe platform heterogeneity, as reported by Grønli et al. [6] and Escoffier and Lalanda [7], developing apps targeting multiple platforms using the native approach could be seen as inherently not feasible. Consequently, adoption of cross-platform mobile development frameworks has been noted [2]. According to Majchrzak et al. [8], such frameworks allow for a single codebase to be deployed to a variety of platforms, with only smaller platform-specific modifications depending on approach and framework. Platform heterogeneity is often handled by the framework, rendering user interface development a one-time job in certain frameworks, as they determine which interface components to render based on the device type and platform on which the app is executed (e.g. [1]).

Cross-platform development has been the target of academic interest since its inception in 2009-2010, with Charland and LeRoux [9] conducting one of the more initial

This paper was presented at the NIK-2017 conference; see http://www.nik.no/.
¹https://ionicframework.com/docs/components/
Due to early and ongoing academic interest, we have been able to identify a well-established body of knowledge for cross-platform mobile development. Research on the subject includes a wide variety of topics, including technical performance studies (e.g. [5]), Human-Computer Interaction (HCI) studies on quality perception and user experience (e.g. [4]), proposed requirements for future development efforts (e.g. [10]), and introductions of novel development approaches (e.g. [11]). Overall, technical implementations are frequently included to provide data and evaluate hypotheses.

Technical artifacts are in fact frequently used in computing and software engineering literature to support claims and provide evidence, also outside of the mobile computing context. Its importance is backed by renowned computer scientist Niklaus Wirth, "The art of designing artifacts to solve intricate problems is the ultimate goal of an academic institution in the field of computing and programming" [12, p. 2]. Thus, while traversing the literature, we identified the lack of a standardized baseline for technical research implementations as a gap to target. Throughout our paper, we explore technical implementations used in existing research on cross-platform mobile app development. We do so in order to identify and answer the following,

- **RQ1**: Which platforms, mobile devices and development approaches are commonly included in cross-platform research evaluating technical implementations (apps);
- **RQ2**: Which device feature APIs are commonly included in evaluations and comparisons in such research.

Based on existing literature, we propose a technical baseline for future research to draw from. We have extracted core details from 14 papers, and use those as the foundation for our proposal. Constructing such a baseline proposal can be of a certain complexity, as cross-platform mobile app development is more a hypernym than a specific technology. Within this term, we find different technical approaches and associated frameworks for architecting and conducting app development (e.g. [8] [13] [14] [15] [16]). Established taxonomy for cross-platform development typically mentions five overarching development approaches to dominate the field (e.g. [3] [17]) which include Hybrid, Interpreted, Cross-Compiled, Model-Driven and Component-Based development. We encourage interested researchers to read such as Heitkötter et al.’s seminal paper, and El-Kassas et al.’s taxonomy study to gain in-depth understanding of the approaches.

For the sake of clarity, we briefly introduce each approach. The Hybrid approach works by having a mobile-friendly website deployed together with a native app’s codebase. By implementing an embedded web browser (WebView) into the native app, the website can be rendered to the screen using the browser widget. Because a native app is the foundation of any Hybrid app, submitting to the different App Stores works as expected.

The Interpreted approach depends on an on-device interpreter to interpret the app’s codebase which in turn renders native user interfaces. Thus, the approach differs greatly from the Hybrid approach which renders web-based interfaces. Nevertheless, a native app is also the foundation of any Interpreted app, and the JavaScript codebase is deployed as part of the native app’s codebase for this approach as well.

The Cross-Compiled approach does not depend on interpreters or embeddable browsers, as it compiles platform-specific binaries from a single codebase. Thus, using a common programming language, native apps for targeted platforms are generated.
The Model-Driven and Component-Based approaches are less adopted by practitioners than the previous approaches, but do generate interest amongst academics according to Umuhoza and Brambilla [18]. In these approaches, native apps are generated based on models and specifications rather than web technologies.

The rest of this paper is structured as follows. Section 2 describes the literature search and selection process, and includes an assessment and review of related work. Our findings are presented in Section 3, and are further elaborated upon during the discussion in Section 4. The baseline is presented throughout Section 5, and we end the paper with a conclusion in Section 6.

2 Literature Search and Selection

To target studies evaluating and drawing from technical implementations, the literature search has been condensed to primarily include papers embracing a design and creation type of methodology, focusing on artifact development and evaluation. We conducted the literature search using IEEE Xplore, Science Direct, ACM Digital Library and Google Scholar. Our search queries were comprised of the following terms: \{Cross-platform development, Cross-platform mobile development, Hybrid app development, multi-platform mobile development\}. Of 50 papers identified as interesting in our context, a total of 14 papers included one or more technical implementations as part of their research effort, thus were assumed directly relevant for review. The remaining papers were to a large extent of a descriptive nature.

Related Work

Do note that no baseline or contribution similar to what we propose has been identified, or even exist to the best of our knowledge. We identified the knowledge gap through numerous extensive previous literature reviews, and found that we could contribute with a more standardized specification for future research to draw from. Consequently, in this section we rather present in short the state of research on cross-platform development. Further assessment of the 14 identified studies is presented in Table 2.

The majority of technical research on cross-platform fits within three categories, Performance studies, HCI and UX studies and Framework Comparison studies. Our baseline as presented in Section 5 is thus based on these. The fundamental papers on cross-platform tend to be framework comparison studies, including such as Heitkötter et al. [3] and Corral et al. [19], mapping approaches and frameworks to requirements and important factors. Also newer research focus on framework-level differences, such as Majchrzak et al. [8], but do tend to draw more from technical assessments, thus help validate the need for this very article.

We have also identified a newfound research interest of analyzing data from the app stores. Such studies help form the foundation of technical baselines. For instance, Ali and Mesbah [2] answer questions such as the prevalence of Hybrid apps in the App Stores by traversing the code of 1.1 million apps, finding the PhoneGap framework to be highly popular – thus its inclusion in our baseline. Alternatively, Mercado et al. [20] analyze the language of more than 780,000 app reviews. Their contribution is of immense value and helps to better understand users’ perception of cross-platform apps on a massive scale.

We will further elaborate on related work by presenting them organized and categorized as a main part of our findings. The next section will take the tour through platforms, approaches, devices, features and perceived complexity.
3 Findings

Our findings have been condensed into the two tables presented below. The order of papers listed in the tables is random, however both tables are ordered identically.

Platforms, Approaches and Devices

Certain abbreviations are used, including WP (Windows Phone), PG (PhoneGap), jQM (jQuery Mobile), ST (Sencha Touch), IAF (Intel App Framework) and RN (React Native).

Table 1: Overview of platforms, approaches and devices

<table>
<thead>
<tr>
<th>Paper</th>
<th>Approaches (Frameworks)</th>
<th>Devices (Platform)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[21]</td>
<td>Native</td>
<td>Sony Xperia Z3 (Android)</td>
</tr>
<tr>
<td></td>
<td>Hybrid (PG)</td>
<td>Nokia Lumia 625 (WP)</td>
</tr>
<tr>
<td>[8]</td>
<td>Hybrid (Ionic)</td>
<td>Nexus 5X (Android)</td>
</tr>
<tr>
<td></td>
<td>Interpreted (RN, Fuse)</td>
<td></td>
</tr>
<tr>
<td>[22]</td>
<td>Web app</td>
<td>Samsung i9250 (Android)</td>
</tr>
<tr>
<td></td>
<td>Hybrid (PG)</td>
<td>Samsung Galaxy S5 (Android)</td>
</tr>
<tr>
<td></td>
<td>Interpreted (Titanium)</td>
<td>Apple iPhone 4, 5 (iOS)</td>
</tr>
<tr>
<td></td>
<td>Cross-compiled (MoSync)</td>
<td></td>
</tr>
<tr>
<td>[5]</td>
<td>Native</td>
<td>Apple iPhone 4, 6 (iOS)</td>
</tr>
<tr>
<td></td>
<td>Hybrid (PG)</td>
<td>Acer Liquid E330 (Android)</td>
</tr>
<tr>
<td></td>
<td>Cross-compiled (Xamarin)</td>
<td>Nexus 6 (Android)</td>
</tr>
<tr>
<td></td>
<td>Interpreted (Titanium)</td>
<td></td>
</tr>
<tr>
<td>[23]</td>
<td>Web app (jQM)</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Hybrid (PG)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cross-compiled (MoSync)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interpreted (Titanium)</td>
<td></td>
</tr>
<tr>
<td>[24]</td>
<td>Native</td>
<td>Samsung GTI-5500 (Android)</td>
</tr>
<tr>
<td></td>
<td>Web app</td>
<td>Apple iPhone 4 (iOS)</td>
</tr>
<tr>
<td></td>
<td>Hybrid (PG)</td>
<td>HTC Desire 4 (Android)</td>
</tr>
<tr>
<td></td>
<td>Interpreted (Titanium)</td>
<td></td>
</tr>
<tr>
<td>[25]</td>
<td>Java2Csharp by Eclipse</td>
<td>WP emulator</td>
</tr>
<tr>
<td></td>
<td>Java2C# by Tangible</td>
<td></td>
</tr>
<tr>
<td>[26]</td>
<td>Native</td>
<td>HTC Nexus One (Android)</td>
</tr>
<tr>
<td></td>
<td>Hybrid (PG)</td>
<td>HTC Magic (Android)</td>
</tr>
<tr>
<td>[27]</td>
<td>JIL</td>
<td>Sony Ericsson Xperia (Android)</td>
</tr>
<tr>
<td>[28]</td>
<td>Hybrid (PG, ST, jQM)</td>
<td>Simulators</td>
</tr>
<tr>
<td></td>
<td>Interpreted (Titanium)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cross-compiled (Xamarin)</td>
<td>Delphi XE6</td>
</tr>
<tr>
<td>[14]</td>
<td>Native</td>
<td>Galaxy Nexus (Android)</td>
</tr>
<tr>
<td></td>
<td>Hybrid (PG + jQM)</td>
<td>Nexus 5 (Android)</td>
</tr>
<tr>
<td>[29]</td>
<td>Hybrid (Ionic, ST 2, jQM, IAF, MGWT, Famou.us)</td>
<td>Apple iPhone 4, 6 (iOS)</td>
</tr>
<tr>
<td></td>
<td>Interpreted (Titanium, Adobe AIR)</td>
<td>Sony Xperia E3 (Android)</td>
</tr>
<tr>
<td></td>
<td>Cross-compiled (Xamarin, NeoMAD)</td>
<td>Nexus 6 (Android)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Nokia Lumia 925 (WP)</td>
</tr>
<tr>
<td>[4]</td>
<td>Native</td>
<td>Longitudinal user study, variety of undisclosed devices.</td>
</tr>
<tr>
<td></td>
<td>Titanium</td>
<td></td>
</tr>
</tbody>
</table>
## Features and Perceived Complexity

Table 2: Overview of features and perceived complexity

<table>
<thead>
<tr>
<th>Paper</th>
<th>Features</th>
<th>Review of Technical Artifact and Study</th>
</tr>
</thead>
</table>
| [21]  | User Interface  
        App Navigation  
        Network Request | Few features included. Artifact is used for quantitative performance measurements, but lack features to properly do so. Authors subjectively discussing the user experience and appearance of the apps. Interesting results to draw from and further empirically evaluate in future research. |
| [8]   | User Interface  
        App Navigation  
        Camera  
        Image Gallery  
        GPS  
        Contact List  
        Network Request | Features chosen in cooperation with industry partner. Authors assess different aspects of app development drawing from implementations using technical frameworks of different cross-platform approaches. Future work would include performance testing the feature-rich implementations. |
| [22]  | Camera  
        GPS  
        Media Access  
        Light Sensor  
        Proximity Sensor  
        Compass | Extensive study including feature-rich apps of different approaches, used to performance test and evaluate differences in hardware utilization and optimization. Research methods should be of interest for future research. We find the overall study to be of great interest and contribution to the field. |
| [5]   | App Navigation | Only one feature included. App navigation used to measure navigation load time. Otherwise an interesting study focusing on measuring CPU, memory and disk space. The lack of features assessed renders the study less comprehensive than its potential. |
| [15]  | Network Request | Provides a set of decision factors and subjective claims. Simple implementation. Tested on only one device, an Archos tablet. Measurement of memory using interesting metrics, CPU and power consumption. Methods can be of future interest. |
| [23]  | User Interface  
        Game Mechanics | Authors merely scratch the surface of studying animations. Focuses on animation API support rather than CPU and battery usage. No mention of Frames-per-Second measurement. Unknown implementation complexity, but is classified by the authors as a "serious game" [23, p. 1]. |
| [24]  | Camera  
        Network Request | Scoring a cross-platform app on a variety of factors, albeit with little empirical evidence. Technical implementation looks non-complex, with complex calculations for image recognition being offloaded onto a backend server through a network request. |
<table>
<thead>
<tr>
<th></th>
<th>User Interface</th>
<th>Results of source-code translation are promising, as their game implementation was successfully translated between the languages. However, this approach requires separate implementations of user interfaces. Also, the implementation is rather non-complex, but further work includes applying the approach to more complex products.</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>User Interface</td>
<td>The technical artifact provides a simple user interface to interact with built-in benchmarking functionality. This approach is interesting in the context of this study being a performance analysis. They include a lengthy array of features, and provide a comparison between a native app and a cross-platform PhoneGap app executing different tasks using the features. This includes measurements such as response time for GPS location requests, file I/O reading and writing and device contact list access.</td>
</tr>
<tr>
<td>27</td>
<td>User Interface</td>
<td>Study evaluates a technical artifact incorporating cooperative game design elements, allowing users to leverage their device’s GPS. App features an especially simple user interface. They list certain approach limitations, but the study could have been further extended through e.g. performance testing the JIL approach.</td>
</tr>
<tr>
<td>28</td>
<td>User Interface</td>
<td>Authors present a review of development approaches, their case study and technical implementations. Study includes three apps to compare frameworks, done mostly from a technical perspective. Artifacts feature simple user interfaces and navigation patterns, without extensive use of device features. Interesting albeit minimal study.</td>
</tr>
<tr>
<td>29</td>
<td>User Interface</td>
<td>Comprehensive study including evaluations of an open source framework-comparison suite. The blend of open source and academia allows for software developed by practitioners to be properly assessed within the rigorous foundation of academic research. Overall a study to draw from in future work.</td>
</tr>
<tr>
<td>4</td>
<td>App Navigation</td>
<td>Highly interesting contribution to the field. Authors present thorough laboratory and longitudinal studies focusing on user perception of cross-platform (versus native) apps. It is one of few studies of its sort. Technical implementation is seemingly complex, albeit containing relatively few device features, it contains enough functionality for regular user engagement.</td>
</tr>
</tbody>
</table>
4 Discussion
Platforms, Devices and Development Approaches

Android was included in 11 studies, making it the most popular platform for conducting research. iOS occurred in six studies, and Windows Phone in four studies. The difference between iOS and Windows Phone occurrence is interesting, as Apple greatly outperforms Microsoft on sales and available apps, according to Statista [30].

Test devices range from cheap low-end phones up to the market-leading ones. We find it less important to give exact device recommendations due to the nature of such devices, but our baseline nevertheless includes a valid overview.

We found that the Native, Hybrid, Interpreted and Cross-Compiled approaches are all commonly included in technical studies. Within each approach, associated frameworks were identified. PhoneGap, either standalone or together with Sencha Touch or jQuery Mobile, was the most-used Hybrid development framework, occurring in nine studies. The Ionic framework, also of the Hybrid approach, was included in two studies.

It might enjoy more presence in future research due to its increasing popularity. For the Interpreted approach, Appcelerator’s Titanium framework was included in six studies, making it the most popular framework in identified research. Also, React Native, Fuse and Adobe AIR had presence although limited, only occurring once each. While Titanium has been the most popular Interpreted solution in previous research, the React Native framework has seen a massive increase in developer popularity in comparison, e.g. using the Github star count: 2242 for Titanium [3] vs. 52333 for React Native [4] (August 2017). We advise academia to make a change of direction in terms of frameworks to keep research relevant also amongst practitioners.

Majchrzak et al. claimed that ”[…] Approaches that originate in the scientific community and are theoretically sound […] are not easily adopted by industry […]” [8, p. 1] Such approaches include Model-Driven and Component-Based development. However, the lack of inclusion of such approaches in the identified literature could suggest that also the scientific community avoid them in studies including technical evaluations. One reason for this could be the lack of a standard modeling language for mobile development using e.g. Model-Driven Development, as further discussed by Umuhoza and Brambilla [18].

Features and Platform APIs

Our second focus was the investigation of features and the platform APIs. There is a great variety of device features available through the different mobile platforms’ Software Development Kits (SDK) (e.g. [31]). We found that in our pool of identified research, inclusion of different features depended on the type of study. For example, the inclusion of navigation between pages in Willocx et al.’s study [5] made it possible to measure and compare navigation- and page load time. The study focused on performance testing different aspects of cross-platform developed apps, such as launch time and hardware utilization. However, in terms of the study’s extensiveness, it could be deemed less comprehensive than such as Ciman and Gaggi’s performance experiment [22], including numerous features and sensors. Had Willocx et al. [5] included a wider array of features, the two studies and their results could be compared, proving or disproving their
hypotheses and suggestions. Nevertheless, they each provide important contributions.

For other types of studies, such as those relating to topics within HCI and User eXperience (UX), the inclusion of user interfaces dominates (e.g. [4, 14, 23]). Issues related to user interfaces are typically raised in studies on cross-platform apps regardless of overarching topic (e.g. [3, 15, 21, 25]). In fact, the look and feel of apps are issues included in studies on requirements for cross-platform frameworks and tools, according to Gaouar et al. [10].

Certain features are considerably more common in research than others. The use of network requests occurs in eight studies, which according to Majchrzak et al. [8] is a feature typically found also in business apps for consuming data from backend APIs. The camera feature is included in three studies, and the purpose of the inclusion varies from comparing implementation complexity (8) to measuring device performance such as battery and memory usage (22). The same is typically true for GPS access and similar sensors, including light, proximity and compass. As the camera is a common feature included in any type of app, being business (8) or social (e.g. Snapchat, Instagram, Facebook), we acknowledge the importance of including it in studies on performance and framework comparisons. We also find inclusion of GPS to be of importance in such studies.

Some papers, such as a study by Biørn-Hansen et al. [16], compare frameworks on feature implementation complexity and availability. The contribution of such papers could be said to increase in a linear fashion for each new feature added to the comparison. As an example, Willocx et al.’s [5] study contributed with interesting insights on cross-platform performance, but its validity and usage for decision making would have been even greater with the inclusion of more features.

5 Proposed Baseline
Approaches and Frameworks

We suggest the following approaches and associated frameworks to be included in future research. The suggestions are a mix of technologies found in the traversed literature as well as up and coming and popular technologies typically discussed by practitioners. We do this in an attempt to push fellow researchers away from deprecated, yet commonly included frameworks (such as MoSync), and instead focus on frameworks receiving attention in developer communities.

• **Hybrid**: PhoneGap and Ionic Framework. Consider Sencha Touch, Quasar Framework, Onsen UI and RhoMobile (Rhodes). Avoid Intel App Framework due to discontinuation.

• **Interpreted**: Titanium Appcelerator and React Native. Consider NativeScript, Weex and Fuse. Consider Tabris.js as a commercial closed-source solution.

• **Cross-compiled**: Xamarin. Consider NeoMAD as a commercial closed-source solution. Avoid MoSync due to discontinuation.

We suggest to consider including the Model-Driven and Component-Based development approaches in studies that could benefit from such, however as previously explored, these approaches are typically not adopted by the industry to the same extent as in academia [8]. We also suggest adventurous researchers to consider inclusion of the Progressive Web Apps approach as discussed by Biørn-Hansen et al. [16].
Devices and Operating Systems

In Table 1 we find that all the major operating systems are included; Android, iOS and Windows Phone (Windows 10 Mobile) are used regardless of type of study. Both real devices and simulators/emulators are commonly used. To optimize research for validity in real-world contexts, we suggest the use of a wide range of devices. Thus, while research labs might be equipped with state of the art equipment, performing tests using lower-end devices should be deemed of utmost importance for validity in emerging markets. We suggest including an array of devices such as (as per 2017)

- **iPhone (iOS):** An older device such as iPhone 4, and a top-range such as iPhone 7.
- **Android:** A low-range device such as the HTC Desire HD or an older Nexus phone, a medium-range device such as the Nexus 5 or Motorola Moto G3/G5, and a top-range such as the Google Pixel or Samsung S8.
- **Windows Phone:** While their marketshare is minimal, inclusion of Windows Phone devices would only increase the level of contribution and the study’s validity. We suggest testing on a low-range device such as the Nokia Lumia 550 or 625, a medium-range device such as the Nokia Lumia 650 and a top-range device such as the Nokia Lumia 925 or 950 (XL).

The inclusion of a wider range of devices and operating systems will inherently lead to a contribution with more validity and purpose for decision making and research. Thus, practitioners and researchers alike should also consider testing on more than just one device within each range category.

Features

We find that features included typically depend on the type of study. Based on our findings from Table 2, we suggest features deemed important for three types or categories of studies:

- **Performance studies:** A wide range of sensors, such as proximity, light, accelerometer and GPS. Inclusion of file I/O access measurement and camera battery drainage, CPU and memory usage. For user-oriented performance studies, include such as animations ([4][23]) and interactive elements and measure Frames-per-Second (should be as close to 60 as possible). Measure navigation time ([5]).
- **HCI and UX studies:** User interfaces following the platforms’ design guidelines. Inclusion of app navigation and navigation patterns. Implementation of interactive interface elements. Inclusion of device features would depend on the type of app being user tested and evaluated.
- **Framework comparison studies:** The more features included, the more comprehensive the study and its contributions are. We suggest, as a minimal baseline, to include app navigation, camera and GPS access, and native feature access through bridges or similar. To extend, include also different sensors as mentioned above.
6 Conclusion and Further Work

Due to differences between categories of studies, no single baseline could fit all types of applied cross-platform research. Thus, in the Features subsection we outlined baselines for performance studies, HCI and UX studies, and framework comparison studies. For the approaches, frameworks and devices, we find that our suggestions are of high relevance regardless of category or type of study. It is evident, and inherently so, that approaches and frameworks will over time deprecate and be replaced by newer technologies. The same is most certainly true also for the availability and support of device features. However, the validity and future-proofing of our contribution lies in the outlined baseline, where newer technologies could be mapped into the sections above.

The studies also varied greatly in perceived complexity. That goes for the papers themselves, their contribution and the technical artifacts included. We acknowledge that each paper contributes with valuable insight to the body of knowledge, but that an increase in complexity of some could increase their contribution significantly.

References


