A Survey and Taxonomy of Core Concepts and Research Challenges in Cross-Platform Mobile Development

ANDREAS BIØRN-HANSEN, Department of Technology, Kristiania University College, Norway and Brunel University, United Kingdom
TOR-MORTEN GRØNLI, Department of Technology, Kristiania University College, Norway
GHEORGHITA GHINEA, Brunel University, United Kingdom

Developing applications targeting mobile devices is a complex task involving numerous options, technologies and trade-offs, much so due to the proliferation and fragmentation of devices and platforms. As a result of this, cross-platform app development has enjoyed the attention of practitioners and academia for the previous decade. Throughout this review, we assess the academic body of knowledge and report on the state of research on the field. We do so with a particular emphasis on core concepts, including those of user experience, device features, performance, and security. Our findings illustrate that the state of research demand for empirical verification of an array of unbacked claims, and that a particular focus on qualitative user-oriented research is essential. Through our outlined taxonomy and state of research overview, we identify research gaps and challenges, and provide numerous suggestions for further research.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Human-centered computing** → **Mobile computing**; **Empirical studies in ubiquitous and mobile computing**;

Additional Key Words and Phrases: Cross-platform mobile development, mobile computing, app development

ACM Reference Format:
Andreas Biørn-Hansen, Tor-Morten Grønli, and Gheorghita Ghinea. 2018. A Survey and Taxonomy of Core Concepts and Research Challenges in Cross-Platform Mobile Development . *ACM Comput. Surv.* 1, 1 (July 2018), 35 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

Applications for mobile platforms have over the last decade been the driving force for the smartphone revolution. The success has spread from smartphones to a variety of devices such as tablets, wearables and sensors, all recognized today as part of the mobile devices platform. Despite the huge success and substantial progress in relation to software platforms, hardware specifications, development methods and use there is still a long way to go to be at a standardized level.

As of 2017, more than five million mobile apps were available throughout different mobile app stores, including Google Play (2.8m), Apple App Store (2.2m) and Windows Store (669k) [88]. Moreover, the two leading app stores have seen massive growth in the number of available apps. Between June 2015 and June 2016, the Apple App Store increased its portfolio with 500 thousand

Authors' addresses: Andreas Biørn-Hansen, Department of Technology, Kristiania University College, Christian Krohgs gate 32, Oslo, 0186, Norway, Brunel University, Kingston Lane, Uxbridge, London, United Kingdom, bioand@westerdals.no; Tor-Morten Grønli, Department of Technology, Kristiania University College, Christian Krohgs gate 32, Oslo, 0185, Norway, tmg@westerdals.no; Gheorghita Ghinea, Brunel University, Kingston Lane, Uxbridge, London, United Kingdom, george.ghinea@brunel.ac.uk.

0360-0300/2018/7-ART $15.00
https://doi.org/0000001.0000001

apps [89]. From February 2016 to March 2017, 800 thousand new apps were introduced into the Google Play Store [91]. These massive software repositories are the results of arduous development efforts from hobbyists, freelancers and enterprise developers alike. The app stores' popularity, for both developers/publishers and consumers, is seemingly ever-increasing. The statistics portal *Statista* forecasts an increase from 149.3 billion app downloads in 2016, to 352.9 billion downloads in 2021 [87]. Revenue from such apps are projected to reach into the hundreds of billions USD in year 2020 [90]. For businesses and individuals wanting to participate in this fast-expanding and ever-growing economy, presence throughout the major app stores is required, in the form of downloadable mobile apps. As we illustrate and discuss through the reminding sections of this review, development of mobile apps is inherently complex due to factors such as vast device and version fragmentation in the Android ecosystem [71], the wide array of technical solutions available for conducting app development – as outlined in Table 1, the rapid proliferation of- and advancements to technical frameworks and tools, and more of the like.

The majority of the apps throughout the app stores [5, 98] are developed using a development technique, or *approach*, commonly referred to as *Native* [49, 85]. This development approach uses the programming languages, tools and development environments specifically meant to produce apps for a given operating system (henceforth referred to as *platform*) [5, 31, 85]. For example, developing *Native* apps for the iOS platform requires knowledge of the development environment Xcode, as well as the programming languages Swift and/or Objective-C [105]. On the other hand, Android apps require knowledge of the development environment Android Studio, as well as Java [105] and/or Kotlin, the latter a recent addition to the otherwise Java-dominant Android development ecosystem [83]. Consequently, apps developed for the Android platform cannot deploy to- or execute on an iOS device [3]. The same is true vice versa. This is due to inherent differences in development and compiling techniques, and overall Application Programming Interface (API) support. Thus, an app that should reach customers across different platforms must be developed from scratch for each new platform supported [52]. The result may be two or three inherently separate codebases for an app that should be available for users across the Android, iOS and Windows Phone platforms. From a maintainability point of view, separate codebases require access to specialized platform knowledge, and e.g. bug fixing might take three times longer with three supported platform compared to only one supported platform [52]. The same would be true for addition of new features, removal of old features, and so on. Due to the popularity of app usage- and the complexity of app development, various technical development techniques and tools have come to either supplement or in part replace the *Native* development approach [62, 80]. The umbrella term for alternative development approaches is usually referred to as either *multi-platform* or *cross-platform* development in industry and academia alike [29, 81], whereas the latter term will be favoured throughout this article due to its prevalence in literature. Cross-platform mobile development is a fast-paced and ever-changing field of research and practice [62]. While the rationale behind adoption of cross-platform approaches may differ, budget, time and knowledge are listed as recurring reasons in literature for avoiding traditional Native mobile development and rather opt for cross-platform development [8, 17, 55]. In a frequently cited study conducted by Xanthopoulos and Xinogalos, the authors state that

> "The ultimate goal of cross-platform mobile app development is to achieve Native app performance and run on as many platforms as possible" [105, p. 1]

Exactly how one may achieve Native-like performance when conducting cross-platform development depends on the *development approach* employed [29]. Different approaches render distinct results and differ in fundamental concepts and practice. The pool of common cross-platform approaches includes *Hybrid*, *Interpreted*, *Cross-compiled* and *Model-Driven* development [13, 101],

with a novel approach coined *Progressive Web Apps* also on the rise [13]. Within each of these approaches, a plethora of technical frameworks exists, facilitating the development of cross-platform apps [19, 62, 67]. In Table 1, we present a condensed and exhaustive list of such technical frameworks, although some of those, including Xamarin, Titanium Appcelerator and PhoneGap (Cordova) are more prevalent in both academic and industry literature. In fact, when discussing the popularity of these frameworks, PhoneGap won the People's Choice award at the Web 2.0 expo in 2009 [25], due to enabling web developers to develop and produce mobile apps using existing web development knowledge [62].

Just as there are distinct differences between the overarching development approaches, each associated framework will differ in popularity, feature accessibility, interface rendering techniques, add-ons, size of community, functionality and more [62]. Ultimately, deciding on a framework depends on a variety of variables, e.g. app feature complexity and integrations, user interface- and user interactivity complexity, availability of support, and licensing terms [49]. Nevertheless, the fragmentation and plethora of available frameworks renders decision making processes complex [58]. Previous research has indeed provided insights on cross-platform frameworks requirements [24, 41] and criteria [50], trying to ease decision making. However, such overarching papers seemingly tend to lack empirical evidence for their statements [49, p. 9], rather focusing on listing important factors and mapping them loosely to technical frameworks.

The purpose and contribution of this survey paper is to provide a thorough overview of the state of research, and through doing so introduce challenges and possibilities for further research, resulting in a taxonomy of core concepts found in this field of research. The rest of this paper is structured as follows. In Section 2, we discuss the five major cross-platform development approaches, and present an exhaustive list of technical frameworks categorized according to their associated approach. Sections 3 through 7 are assigned discussions and analyses of the overarching concepts traversed in this review, specifically we start by outlining the research foundation, then proceed to discuss User Experience, Software Platform Features, Performance and Hardware, and finally Security. Following these, in Section 8, we provide a taxonomy and overview of the state of research on cross-platform development. In Section 9, we conclude the review by highlighting and discussing future directions of cross-platform development research, along with suggestions for further research building on our findings and the derived taxonomy.

2 DEVELOPMENT APPROACHES AND DOMAIN TERMINOLOGY

The introductory section briefly mentioned a variety of overarching approaches for conducting cross-platform mobile development. Most of these approaches are frequently mentioned in the literature (e.g. [29, 49, 85]), and taxonomy should be deemed important for a consistent language when discussing both with academics and practitioners. Each approach, as we further elaborate on below, has their own set of characteristics [29]. This is visually represented in Figure 7, the taxonomy model. Technical frameworks for app development can normally be placed into just one approach, exactly which one depends on such characteristics. This will be further explored both in Table 1, and throughout the remainder of this section.

The coming sub-sections introduce the most prevalent cross-platform approaches and associated frameworks, discussing each to a degree where a fundamental understanding of the benefits and challenges they pose should be conceivable. The naming of these approaches tend to vary between authors and studies, but those used throughout the survey are those most frequently encountered when compared to alternative names, e.g. *"{Native|WebView|Web-to-native} wrapper"* which is sometimes seen in-place of *Hybrid* [77, 102]. This is also true for the categorization of frameworks, i.e. to which approach a framework belong. An example of this is the NeoMAD framework, which by Ettifouri *et al.* [33] is listed under the cross-compiled approach, while Willocx

et al. [102] argue that it is a source-code translator, the latter an approach we have not covered in this survey as it is not commonly encountered in the literature to the best of our knowledge. We also find that one framework, specifically MoSync, can be placed into two approaches, which before it was discontinued [69] provided both a C++ based Cross-compiled framework, in addition to a JavaScript-based framework for Interpreted app development. Each approach pose benefits and challenges, and no single approach is inherently *"best suited for all situations"*, similar to any tool with a particular purpose. It is important to note that *cross-platform* is an umbrella term for a wide variety of concepts, technologies, approaches, frameworks and libraries. The term is also somewhat context-dependent, meaning that cross-platform development could refer to the development of software across multiple device types, not only mobile, as discussed and conceptualized by Rieger and Majchrzak [81]. Nevertheless, throughout this current literature review, we focus on mobile smartphones, leaving other smart devices such as cars, smart homes and Internet of Things devices out of the scope.

We acknowledge that there exist development approaches not included in this review, which have been left out due to lack of prevalence in the literature, and have not been identified as mentioned in industry or practitioner outlets. Examples of such approaches include Component-Based development (e.g. [32, 74]) and Integrated Cross-Platform Mobile Development [28], the latter a proposed approach drawing from best practices inherited from other approaches. However, regardless of their novelty or possible application, we have yet to identify studies and experiments including these approaches, with the exception of those previously cited. Thus, not enough data has been generated to sufficiently or with purpose provide an overview of their state of research.

In the context of the approaches included in this review, we have condensed an exhaustive list of identified technical frameworks, each categorized within an approach that best describe their characteristics. This categorized list of frameworks can be found in Table 1 below. An important note, in the table column *Hybrid*, we mark (*) the frameworks using Cordova as their underlying technology allowing Web-based code to be built for- and executed on device. As the table depicts, the majority of Hybrid cross-platform frameworks do in fact rely on Cordova for these tasks, although some outliers exist, namely Capacitor, Trigger.io, and possibly Kony although we were unable to verify the latter. Cordova's prevalent position is further examined as part of Section 2.1, in which we discuss the Hybrid approach. Furthermore in the context of Table 1, we acknowledge that it contains frameworks of all statuses, ranging from alpha-stage, through production-ready, to having been discontinued for some time. The purpose of Table 1 is to exhaustively introduce the frameworks we have identified in existing research, in industry outlets, and through exploring the field, i.e. regardless of whether a framework is in active development or not.

Table 1. An Overview of Development Approaches and an Exhaustive List of Associated Technical Frameworks

	Development Approaches				
	Hybrid	Interpreted	Cross-compiled	Model-Driven	Progressive Web Apps
Described in	Section 2.1	Section 2.2	Section 2.3	Section 2.4	Section 2.5
Technical Frameworks	Cordova * Capacitor PhoneGap * Ionic Framework * Onsen UI * Framework7 * AppGyver * Quasar Framework * Sencha Touch * Intel App Framework * Intel XDK * RhoMobile Kony ? EvoThings * NSB/AppStudio * Cocoon * Trigger.io	React Native NativeScript Tabris.js Fusetools MoSync Titanium Appcelerator Adobe AIR Smartface Cloud Weex Kony Jasonette LuaView	Xamarin Flutter Codename One Xojo Mobile MonoCross Corona Apportable Qt Mobile MoSync RAD Studio (Delphi) Crosslight RoboVM Marmalade Rhodes DragonRAD	MD^2 MobML Applause Mobl Mendix Appian MAML Mobia Modeler Xmob AXIOM MOPPET mdsl Automobile WebRatio XIS-Mobile MobDSL	Ionic Framework Zuix Mithril Polymer Svelte Preact Vue.js Angular React.js Stencil.js Glimmer.js Ember.js viperHTML Moon.js

2.1 The Hybrid Approach

This approach allows for the use of regular web technologies including HTML, CSS and JavaScript to be leveraged in an app development context for implementation of user interfaces and business logic. The approach works internally by initializing a new Native app project, which includes a WebView component [59] as well as code for communication between the WebView and Native code [7, 43]. As the WebView component is, simply put, an embeddable web browser [7], it allows for the execution and rendering of HTML, CSS and JavaScript files. These files make up the app's logic and user interface, and are included as part of the app bundle together with the Native app project and code. The developer can then point the WebView to render a specific HTML page, programmatically controlling what the component displays to the user [43]. Accordingly, as an Hybrid app developer, one will write the entirety of the front-end and business logic of the app using web technologies, then have a regular Native app wrap and bundle the web code and displaying it through the embedded WebView component. Due to this technique, the Hybrid approach is also referred to as *Native-wrapper* [101], as it wraps web assets into a publishable, deployable Native app, installable from any typical mobile app store.

As this requires a fair bit of code and setup, Apache Cordova has become a popular tool and library for initializing new Hybrid apps [101]. Instead of leaving all the above work to the developer, i.e. the initialization and setup of WebViews and communication protocols, Cordova will through the use of a command-line tool generate a new Native app including a WebView and two-way communication between the WebView and Native code [21]. Such communication, referred to as *bridging* [1], allows developers to communicate with platform-specific Native code from within a non-native environment, such as a JavaScript context. For reference, this technique of executing code between distinct programming languages is also called Foreign Function Interface (FFI). One would typically use bridging, or FFIs, to e.g. process or handle tasks deemed too expensive for JavaScript, or to leverage functionality typically only accessible in Native environments [1, 80]. The bridge is called from the app's JavaScript code, executing a Native-code function and potentially returning some value from the Native-side back to the JavaScript context, e.g. a GPS coordinate requested from JavaScript that must be fetched from Native code. Thus, bridging functionality should be deemed of high importance to ensure a Native app-like user experience.

In addition to providing easy Hybrid app initialization, Cordova also provides a plugin system with thousands of available plugins, including such as camera access, GPS access and contact list access, features requiring the aforementioned bridging system to function [23]. The plugin system also provide a standardized method for the Hybrid app developer community to contribute with additional plugins [11], at least for the Hybrid developer frameworks building on Cordova. Since the Cordova library only provides the foundation for (most) of the Hybrid app development frameworks (ref. Table 1), additional tools and libraries should be used to develop Native-like and Native-feeling user interfaces and interactions [59]. A wide range of such libraries have been identified through previous research and search engine queries. Examples include Ionic Framework[1], Framework7[2], Onsen UI[3] and Sencha Touch[4]. They all have in common the focus of facilitating development of user interfaces for Hybrid apps (e.g. [59]). Because an Hybrid app is a website presented inside a Native app through the use of a WebView component, the user interface may look as Native-app-like or non-Native-app-like as one may wish [45]. However, developing Native-app-like user interfaces which adhere to the interface guidelines of all supported platforms, e.g. Android Material Design

[1] http://ionicframework.com/
[2] https://framework7.io/
[3] https://onsen.io/
[4] https://www.sencha.com/products/touch/

and Apple Human Interface Guidelines, may be challenging and time-consuming to do from scratch [43, p. 7]. This is the power of user interface libraries such as those mentioned above, as they mimick the look of Native app user interface components using HTML, CSS and JavaScript [47].

Because the files all contain standard web technology code, they work in every (embeddable) browser [1] and allow for re-use of existing knowledge for web developers, this approach to app development is highly popular amongst cross-platform developers [5, 67]. An Hybrid app, because of having a Native app as its foundation, can be submitted into app stores in the same way as an app developed using the Native development approach, and thus differs greatly from a regular website or web app [49] which for now is constrained to the browser.

The illustration below provides an overview of how the Hybrid approach works, where Cordova is the controlling entity, managing the execution, rendering and packaging of the app's user interface (HTML, CSS) and the business logic (JavaScript). As we discuss in Section 9.2.1, an alternative to Cordova, named Capacitor, is being developed by the Ionic Framework team. Thus, although the below illustration accurately depict the state of the art in Hybrid app development at the time of writing this review, certain frameworks, thereof Ionic, are likely to incorporate Capacitor at some point in the future.

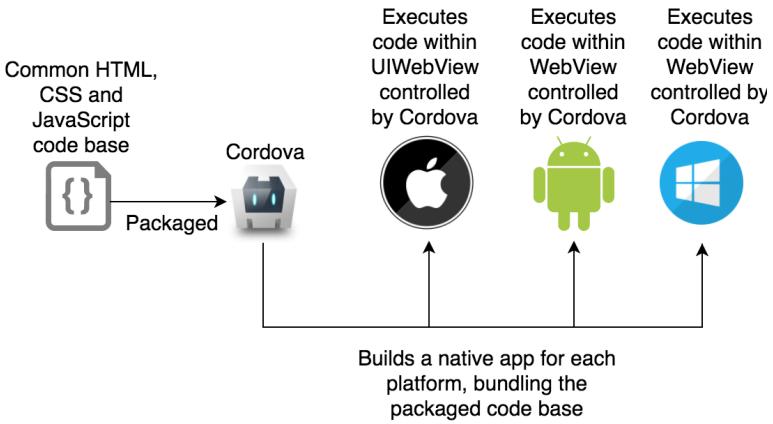

Fig. 1. Overview of the Hybrid approach build workflow

2.2 The Interpreted Approach

Similar to the Hybrid approach described above, it is common to find development frameworks of the Interpreted approach enabling developers to build their apps using the JavaScript language (e.g. [36, 92]), although JavaScript is not an inherent language to the Interpreted approach. Apps developed using the Interpreted approach are fundamentally different from Hybrid apps, as Interpreted apps do not rely on a WebView component to render a bundled website [29]. Instead, Interpreted apps can render actual Native user interface components to the screen, not HTML- and CSS-based views [27], although there are examples of Interpreted tools do not have this as a goal (e.g. the Unity game engine). This is enabled through the use of on-device JavaScript interpreters [62], hence the naming of the approach. In terms of code interpreters, JavaScriptCore is the default interpreter on iOS devices [4, 35]. On Android devices, the interpreter in use differs between frameworks belonging to the approach, but V8 is a frequently used engine (e.g. [4, 35]).

Certain frameworks of the Interpreted approach, such as Titanium Appcelerator, are occasionally incorrectly associated with the Cross-compiled approach (e.g. [32]). While both approaches generate

Native user interfaces, the Interpreted approach does not compile, convert or transpile the codebase into Native byte code, which is how the Cross-compiled approach works. Indeed, as documented by the Titanium framework, a JavaScript interpreter is required as a layer of abstraction [9], making Titanium a framework of the Interpreted approach.

In order to communicate between the JavaScript layer and Native code layer which has access to Native device features, the Interpreted approach also employs the technique of bridging (Foreign Function Interfaces), similar to how the Hybrid approach facilitate such access [11]. An interesting note is that a *"Cordova for Interpreted apps"* has yet to be identified, meaning frameworks of the Interpreted approach lacks the common foundation found in Hybrid apps. Thus, plugins or modules for exposing certain features to JavaScript from the Native environments belonging to one Interpreted framework would not work out of the box in another framework due to differences in implementation and bridging APIs. Examples of such inoperability includes plugins in technical frameworks such as Facebook's React Native and Telerik's NativeScript. While they both belong to the same development approach, the underlying framework APIs are of such different nature that a plugin or module developed for one framework cannot currently work in the other. This in turn fragments frameworks and developer communities of the Interpreted approach greatly, whereas in Hybrid development, all frameworks building on top of Cordova could in theory use the same Cordova plugins.

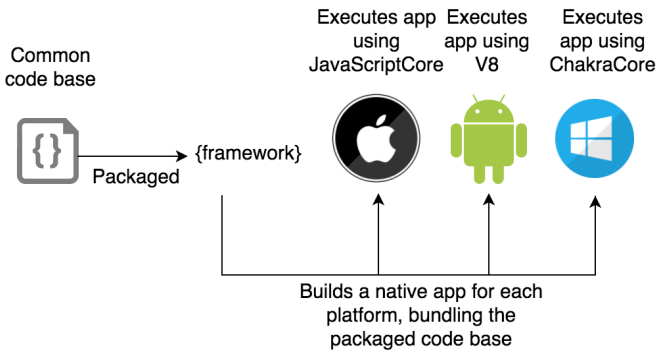

Fig. 2. Overview of the Interpreted approach build workflow

2.3 The Cross-compiled Approach

The core difference between the Cross-compiled approach and the aforementioned approaches, is that due to being compilers, frameworks and development tools of the Cross-compiled approach do not rely on WebView components or on-device (JavaScript) interpreters for rendering of user interface or communication with platform and device features. Instead, a common language such as C# (Xamarin) is compiled to Native byte code executable on targeted platforms [19]. Thus, the bridging layer known from e.g. the Interpreted and Hybrid approaches does not exist in Cross-compiled apps. Neither the use of- nor the access to Native device features is controlled by such a layer, but is rather exposed to the app developer through the framework Software Development Kit (SDK), which accordingly maps functionality to the underlying platforms' SDKs. Another positive consequence of compiling to Native byte code is that of the generated user interfaces, specifically how they are rendered as Native interface components [101].

We identified a lack of consensus regarding which frameworks belong in the Cross-compiled approach. An example of disagreement include a study by El-Kassas *et al.* [29], claiming that the Xamarin cross-platform tool belongs to the *Interpreted approach*, together with e.g. Titanium Appcelerator. On the other hand, Willocx *et al.* [101] claim Xamarin belongs to the Cross-compiled approach due to how it does in fact not rely on interpreters, as it rather compiles a common language into Native byte code [103]. As such, we consider Xamarin a tool of the Cross-compiled approach, due to Willocx *et al.* 's reasoning.

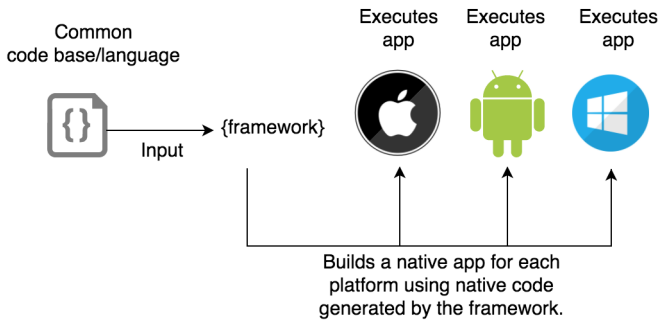

Fig. 3. Overview of the Cross-compiled approach build workflow

2.4 The Model-Driven Approach

This approach draws from the common software development paradigm *Model-Driven Development* (MDD), also referred to as *Model-Driven Software Development* (MDSD) [52]. Note, the first term is favoured throughout this text due to its prevalence in the literature. While being a somewhat commonplace approach in the relevant body of knowledge (e.g. [52, 56, 78, 79]), technical implementations building on the MDD approach are rare among practitioners and in developer communities, also beyond the context of (cross-platform) mobile development [44]. This is further partially confirmed by Umuhoza and Brambilla [93] in their survey on MDD approaches for cross-platform mobile development. They categorize technical tools and frameworks into *Research Approaches* and *Commercial Solutions*, where the latter category only lists four tools, none of which have been frequently cited in pervious literature to the best of our knowledge.

Frameworks of the MDD approach differ in terms of integrated functionality, as discussed by Heitkötter and Majchrzak [51]. The technical framework MD^2, as presented in their study, does not require any knowledge of platform-specific programming languages. According to Ribeiro and Rodrigues da Silva [77] in their study on cross-platform approaches, these types of abstractions are part of the underlying methodology of MDD development. Frameworks of the MDD approach facilitate generation of user interfaces and business logic based on constructed models and templates, suitable for mobile app development as they "[...] allow[s] platform independent modeling, which can later on be transformed to multiple mobile platforms", as described by Usman *et al.* [96, p. 2].

Common for MDD frameworks is a Domain-Specific Language (DSL) [105]. Developers and non-developers alike are enabled by the framework-provided DSL to build their software. Thus, developing apps across mobile platforms will require knowledge of the DSL rather than Objective-C and Java. Generators will then convert the models/code into Native source code for the targeted platforms [51]. From a developer perspective, most MDD frameworks develop and expose their own distinct DSL [93], as illustrated in Le Goaer and Waltham's study titled *"Yet Another DSL for*

Cross-platforms Mobile Development" [60] in which the authors propose the *Xmob* DSL for mobile development. The distinct nature of the DSLs may possibly render knowledge transferability from one MDD-based framework to another low or non-existent. However, MDD framework super-users with expert knowledge of the DSL may develop truly Native apps using shared codebases [51]. In fact, one of the philosophies behind the Model-Driven approach is enabling non-developers and non-technical users with domain expertise to model line-of-business apps based on a provided DSL – being textual or graphical [79].

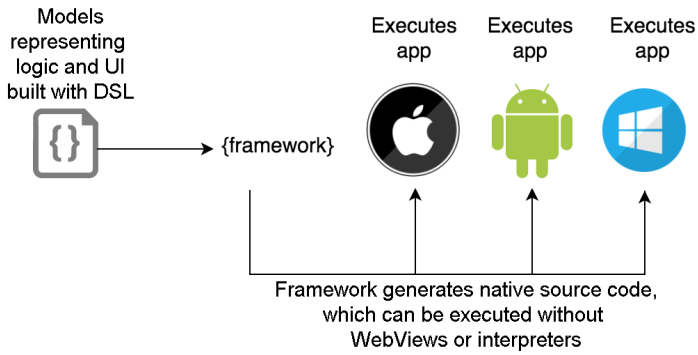

Fig. 4. Overview of the Model-Driven Development approach build workflow

2.5 The Progressive Web Apps Approach

While still lacking some inherent characteristics of other cross-platform approaches (e.g. access to most if not all of platform and device features [13]), Progressive Web Apps - or PWAs for short - have increasingly gained popularity amongst practitioners since 2016 [13]. At its core, a PWA is a web app with enhanced capabilities. While being hosted on- and served by a webserver to users accessing the website's URL in a browser, one goal of this novel approach is to allow for web apps to look and feel like a regular Native or cross-platform built app. Due to being web-based, the user interface of PWAs can be designed to look and feel similar to Native apps using the same methods as in the Hybrid approach, specifically through the use of HTML and CSS for structure and style.

When accessing a PWA-enabled website, a banner will prompt the user asking them to install the website onto their phone. This will download all necessary assets, including JavaScript files, HTML and CSS, images and fonts, and allow for offline usage of the website. The now-offline PWA has also been added to the user's home screen similar to any app downloaded from app stores. In Apple's iOS version 11.3, the underlying technology enabling PWAs to function, i.e. Service Workers, allowing also iOS users to take advantage of Progressive Web Apps, although with a some technical limitations, e.g. lack of state management between sessions, and white screens in the App Switcher, according to Firtman [38]. Upon launching the PWA from the home screen, an artifact-less browser window will open the previously downloaded assets, and read a manifest file to configure a splash screen, icons and colour configurations. In this context, being artifact-less means no address bar, settings icon or similar is displayed, only the website's content – effectively meaning that unless the user knows about the concepts of PWA, they will not know that the app is running within a browser. This greatly increases the app-like feeling of using a PWA compared to a regular website browsed in a traditional way.

Extending a regular web app into a Progressive Web App involves the integration of Service Workers, a JSON-based manifest file and a bundle of static user interface components not dependant

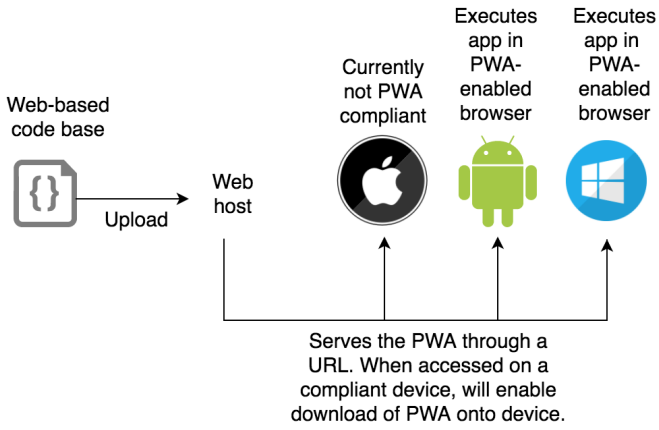

Fig. 5. Overview of the Progressive Web Apps approach build workflow

on dynamic content referred to as Application Shell [13]. The purpose of the Service Worker is to control and manage the app's lifecycle, business logic for data synchronization and handling of push notifications, all through a script file written in JavaScript that is able to execute in the background [42].

The academic body of knowledge on Progressive Web Apps is limited to only a handful of published works (e.g. [13, 63, 65]). These publications, along with practitioners' outlets such as Google Web Fundamentals and the Google I/O conference would indeed act as the foundation for further scholarly research. Along with the advent of Progressive Web Apps, research involving technical assessments, conceptual discussions and case studies would be a natural step towards including PWAs in computing research, e.g. within the field of cross-platform (mobile) development.

3 THE RESEARCH FOUNDATION

While a solid theoretical foundation on cross-platform mobile development has been identified, new development frameworks and approaches seeking to e.g. bring Native user experiences to cross-platform apps have emerged and rapidly gained popularity over the last few years [13]. When Dalmasso *et al.* [24] concluded that cross-platform apps were "Not as good as Native apps" (p. 1) in terms of user experience, and that the quality of such apps range between "Medium to low" (p. 1), these results, due to the age of the study and the dynamic nature of the field, might not reflect the current state of the art. In Section 4, we further explore and discuss studies related to the the users' perspective, as results from newer research has the potential to contradict Dalmasso *et al.* 's findings. This is a testament to the fast-paced innovation within this field of practice and research.

The survey at hand is not the first taxonomy or survey on mobile application development, nor is it hopefully the last. A recent study by Rieger and Majchrzak [81] explore the vast landscape of app-enabled devices, in which they generate a taxonomy based on dimensions related to media input, output, and device mobility. The nature of their taxonomy is distinctly different from that of ours. While their taxonomy is based on a categorization of all app-enabled hardware based on the aforementioned dimensions, our focus is investigating the state of research on cross-platform app development and report of areas within our field in need of further studying. Nevertheless, there most certainly is an overarching topical link between the taxonomies, being that of *apps*. Another taxonomy for cross-platform mobile development was also identified, being that by El-Kassas *et al.*

[29]. The authors describe and illustrate low-level differences between development approaches in a technical matter, thus a study recommended for those interested in a taxonomy of a more software engineering-oriented fashion compared to the study at hand. Thus, whereas [29] provides e.g. code-supported pros and cons of an array of development approaches, as previously mentioned, our aim is rather that of providing an overview of the state of the art and research.

Several comprehensive studies have been identified as part of the literature search process. One example thereof is Heitkötter *et al.* [50]'s seminal paper on cross-platform approach evaluation. Published in 2012, the article is from the early days of cross-platform *mobile development* research. They present 14 criteria for different perspectives of the development process, and proceed to map their results and findings to said criteria. They go into topics such as licensing, platform support, user experience, maintainability and ease of development. The article is one of the most well-cited (165) studies identified. They presented possibilities for future work such as empirically verifying their results and ensuring that academia stays up to date with the technological progress in the field. The latter suggestion has been followed up by academia to a varying degree. While cross-platform development is still actively researched, newer studies tend to perhaps draw too much from previous research rather than including innovative technologies in new experiments. This is especially true for studies of the software engineering type, where older, or even outdated technical frameworks are still scrutinized (e.g. [3, 19, 41]) in favour of newer and less-researched ones [13].

Due to the overarching and all-covering nature of the seminal Heitkötter *et al.* [50] paper, it does not go in-depth into any of its presented topics or evaluation criteria. The authors merely scratch the surface of what is presented, but they lay a foundation for what is important when researching the field. The topics presented could all be of relevance for in-depth research, and the value of the contribution increases accordingly. This paper is also cited in the majority of relevant papers identified. Due to its impact in the research field, a revised version was published by Rieger and Majchrzak [80] seeking to extend the "cross-platform" term into other contexts such as smart homes, smart cars and smart TVs. By doing so, they have established a foundation for research on cutting-edge novel technologies and use-cases. Nevertheless, researching the suggested contexts is a task for future research, as it extends beyond the scope and purpose of this thesis.

Furthermore, a variety of studies are from the earlier days of cross-platform research. Perhaps one of the earliest works identified was that of Charland and Leroux [17]. This is an interesting contribution, as both authors were involved in the creation of the PhoneGap framework during its early days. Their perspectives may thus differ from practitioners and academia. In their paper, Charland and Leroux discuss differences in how Native and web apps were developed in 2011. An interesting observation is that very little has changed since the paper was published. While development tooling and environments have inherently progressed and become more advanced, web apps are still developed using the same languages (HTML, CSS and JavaScript), Native apps are still developed the same way – and still account for most apps in the app stores [98], platform- and device fragmentation (especially on Android) is still very much true [46, 101], and platform conventions (design, user experience) is still a widely discussed subject when debating Native versus cross-platform development [8, 68]. This, in turn, makes for possible future research contributions of great impact. The fact that problem areas identified in the beginning of cross-platform development are still very much tangible and real today, shows that not enough research effort has gone towards developing actual solutions. In the coming section, we discuss one such frequently discussed problem area, specifically user experience and perception quality of cross-platform developed apps.

4 USER EXPERIENCE

Are developers biased in their decisions on cross-platform versus Native development? Will their decisions inherently be coloured by the fact that they work against a goal of well-optimized and well-performing apps and user interfaces, regardless of actual user perception? Can end-users notice the difference between cross-platform and Native apps, and if they do, does it matter? These questions are interesting in our context, as it is commonplace in both academia and industry to find critical and often unbacked claims regarding user experience in cross-platform apps, even regardless of overarching research questions. An example of such is a statement from a well-cited (91) study by Dalmasso *et al.* , reporting that when it comes to the development of user interfaces, cross-platform frameworks and tools cannot provide interface elements comparable to those found in Native apps [24, p. 324]. They also follow up with a statement on the importance of a rich user interface, and that cross-platform frameworks should indeed implement such.

A newer (2015) study by Boushehrinejadmoradi *et al.* also claim that the Hybrid approach and its associated frameworks will generate apps that are inherently subpar compared to Native apps for certain types app categories, but they do not provide any empirical evidence to back their claim [14, p. 441]. Similar claims are also found in cross-platform approach decision frameworks, thereof studies by Latif *et al.* and Lachgar and Abdali [58], reporting that user interfaces generated by the Hybrid approach are subpar to those in Native apps due to execution in the WebView [59, p. n/a] – although no empirical evidence thereof is provided.

A reader with little experience assessing scientific material, or similarly an unexperienced decision-maker may not question the truth of these statements due to being published as peer-reviewed scientific papers. A holistic perspective, possibly breaking this circle of misconceptions would require research efforts toward testing and validating these claims. Alas, literature attempting to confirm or disprove such statements has been found to be – at best – sparse.

The few studies identified focusing on end-user perception of cross-platform apps employ mostly quantitative methods for data gathering. While one study relies on data from laboratory and longitudinal research methods [8], others focus on mining and analyzing quantitative data in the form of app store reviews [5, 66, 68], some through the use of Natural-Language Processing [68]. In fact, these cited studies make up the identified body of knowledge on research on the *user's perspective* of cross-platform apps. Thus, it is evident that a topic this commonly discussed requires additional research and perhaps a wider array of research methods, such as use of equipment including eye trackers and brain-computer interfaces.

The studies incorporating mining and analyzing of app store reviews do display interesting quantitative results, but alas such studies are far between. Nevertheless, a study of this kind has been conducted by Ali and Mesbah [5], presenting a rating scale named *Aggregated User-perceived Rating (AUR)* to attempt to rate user perception based on an app's star rating and number of reviews. They find that the much-criticised Hybrid approach, in this paper represented by the PhoneGap framework, scores the highest on the AUR scale (higher is better), compared to the Titanium Appcelerator and Adobe AIR frameworks, both of which are frameworks of the Interpreted approach. The authors also found that within certain app categories such as comics, business, entertainment and finance, Hybrid apps received a greater AUR than Native apps in the same categories. The authors conclude their study by stating that Hybrid apps can in fact be of such quality that they can provide the same experience as Native apps [5, p. 54].

Their results concur with Malavolta *et al.* 's study [66] on end-user perception of Google Play Store (Android) apps, following much of the same research approach and method for measuring perceived app performance. Indeed, cross-platform apps published to some categories constantly score higher than Native apps in the same categories – e.g. in categories such as business, medical

and lifestyle. These findings also match those presented in [5]. However, both the Entertainment and Music & Video categories vary greatly between the studies. In [5], both categories are found to contain better Hybrid apps than Native ones. This is in stark contrast with results published by Malavolta *et al.* [66]. One explanation could be that there were great variations in the data sets. A total of 3041315 reviews from 11917 apps were the basis of the [66] study, while 9948 Hybrid and Native apps, totalling 19896 apps, were analyzed in [5]. Another explanation for the differences between the studies could be the formula used to calculate rating.

Using Natural-Language Processing to analyze app reviews from Google Play Store and Apple's App Store, Mercado *et al.* 's [68] study on development approach choice's impact on end-user experience fills an important space in the knowledge body. It complements the previous studies ([5, 66]) with a language processing strategy and model rather than formulas for star reviews and review counts. Their findings indicate that end-users perceive apps developed in different cross-platform approaches, differently, and that users leave reviews regarding their varying experiences with such apps [68, p. 48]. It would be of interest to know if the apps receiving negative user reviews belong in either of the outlier categories as discussed by Ali [5] and Malavolta [66]. From a technical perspective, conducting code reviews of the mentioned Hybrid apps identified by Mercado [68] could help answer *why* the apps were rated so poorly. No academic literature has been identified on code-wise optimization of Hybrid apps, thus looking to industry literature makes ever more sense. Books such as *High Performance Mobile Web* by Firtman [37] could help in creating a code review baseline for both Progressive Web Apps and apps of the Hybrid approach. Nevertheless, the findings quoted above are highly relevant for academia and relevant to industry, and fill a gap in which much of the previously published work discuss unbacked claims.

From the qualitative research space, only one study of scale directly involving end-users has been identified. A study by Angulo and Ferre [8] involves both laboratory and longitudinal studies on end-user perception of cross-platform app quality. To rigorously evaluate and compare such quality, the researchers developed four apps, including two Interpreted-approach (one for Android and one for iOS) as well a Native app for each platform to understand the baseline quality expected. Method-wise, they employed the System Usability Scale for their questionnaire, a tool considered an industry standard [95]. Results indicated more scepticism amongst iOS users towards non-native apps than what was found among Android users [8, p. 7]. In their tests, 91% of Android users and 79% of iOS users found the cross-platform app to behave as- or similar to the Native baseline app. This is the only study found that leverage not only quantitative methods (e.g. app review analysis), but focus more on user involvement. Their conclusion, as cited below, helps in disconfirming previous studies' unbacked claims regarding user experience in cross-platform apps, e.g. [24].

> "a good level of UX can be obtained if the cross-platform development framework is chosen carefully in terms of providing adapted interaction styles for each platform, and the development team has UX expertise. But there are more possibilities of getting a better UX by maintaining the control over interaction issues that provides the development of an app with Native code." [8, p. 8]

Alas, the study did not include any cross-platform apps of the Hybrid, Cross-compiled or Model-Driven approaches, the former being more commonly criticized than the others due to depending on the WebView component (e.g. [14, 59]). Nevertheless, this only confirms the need for more user-oriented research on cross-platform apps. A study that does assess the user experience of a Hybrid PhoneGap app, although with a limited group of subjects, is the Heitkötter *et al.* [49] paper. While they have not conducted any user experiments to objectively scrutinize their implementation, the author team discuss their own experience using the Hybrid app, reporting that they found the app to be as responsive as- and to provide comparable performance to a Native application [49, p.

304]. Albeit the possibilities of bias in their reporting, further evaluating similar implementations – though with a subjective group of test subjects – can help in clarifying how users actually perceive cross-platform apps.

A particularly important yet infrequently encountered sub-topic within cross-platform apps and user experience, is the use of apps for those with disabilities hindering regular interaction with a device and its software. In the academic literature on cross-platform app development, discussions on- and development of accessible user interfaces is rarely a subject of focus. The combination of fragmented research efforts along with the introduction of stricter laws and policies on accessibility (non-discrimination laws) [99] leaves a massive gap in the body of knowledge. Universal access to- and accessible design of mobile applications should be considered not only a necessity due to policies and law, but also an important societal deed to avoid discrimination of those with disabilities hindering normal use of mobile devices. The identified lack of relevant literature, both in academic and industry, could be an indication that accessibility, alas, is in general not a target of focus in neither research or practice.

The W3C's *Web Content Accessibility Guidelines*, or WCAG[5] for short, is the overarching and standard documentation and specification for implementation of accessibility measures in the context of both web and mobile [73] – W3C notes that *"there are no separate guidelines for mobile accessibility"* [100]. The WCAG guidelines are commonly adopted by governments for regulations such as the European Accessibility Act [34], and should thus be treated as the baseline and point of departure for future accessibility research.

From the academic knowledge base, we identified a study by Krainz *et al.* [56] to be one of mere few papers covering cross-platform accessibility in any level of detail. The authors study accessibility in the context of the Model-Driven Development approach, alas resulting in both topics effectively competing for the paper's main focus. Their findings lack empirical evidence and assessment of proper accessibility as no actual user testing was conducted on their technical implementation (app). Nevertheless, the implemented app do build on what they refer to as *"best practices and common standards"* [56, p. 2]. User testing commonly includes equipment such as screen readers and Braille readers [82], and voice recognition and eye-tracking [75] hardware and software.

If one is to extend our context-depending definition of mobile cross-platform, being apps that can be deployed to multiple platforms with little to no platform-specific changes, also additional devices (platforms) may be subject to accessibility research. This extension of our definition is proposed by Rieger and Majchrzak [81], and includes such as smart TVs, smart cars (e.g. Tesla) and similar. The W3C WAI working group also include such devices in their work, in which they explain that whether being *"phones, tablets, TVs, and more"* [100], the end-goal is to enable those with disabilities to use apps and the Web.

4.1 Summary of section on User Experience

In this section, we have discussed a wide range of studies and methods related to research on User Experience, and we find that no real conclusions can be drawn from the literature. From studies stating no performance loss identified [49], to laboratory and longitudinal user-oriented studies reporting of mixed results [8], all the way to user review analysis stating that certain types of apps could be seen as unfit for current cross-platform frameworks [68]. It is also evident that further research on accessibility in the context of cross-platform apps should be of monumental importance going forward.

[5]https://www.w3.org/WAI/intro/wcag

Table 2. Summarizing research on User Experience

Approach	Description	Frameworks encountered
Hybrid	We find that the topic of User Experience in Hybrid-based applications is much-debated in the literature. Several quantitative experiments report on the popularity and success of Hybrid apps in the app stores, although they seem more prone to complaints than Native apps. From surveying the literature, we identify a gap of qualitative studies involving the Hybrid approach.	PhoneGap, jQuery Mobile, Sencha Touch, Ionic
Interpreted	The Interpreted approach was scrutinized in the only qualitative User Experience experiment identified. Findings indicate that a penalty in user experience should be expected, especially noticed by iOS users. However, newer research is much-needed to report on the current status-quo, as no newer User Experience studies on the Interpreted approach has been identified.	Titanium, Adobe AIR
Cross-compiled	No studies identified evaluating the user experience in apps generated by frameworks of the approach, although it is commonly included in comparative studies.	-
Model-Driven	One study on accessibility was identified, but no user-oriented experiments were conducted. In terms of research on general user experience, none were identified.	JAXB toolchain
Progressive Web Apps	While we did not identify any user experience-related studies directly, further research could build on the studies carried out by UX/HCI researchers on both mobile web and mobile apps, as PWAs are in their intersection.	-

5 SOFTWARE PLATFORM FEATURES

In their seminal work, Heitkötter *et al.* [50] state that device and platform feature access would typically include camera, GPS, push notifications and similar functionality belonging either to the platform itself (Android or iOS operating system) or the devices' hardware, provided access to through the platform SDKs. Having programmatic access to device and platform features is a

requirement typically listed as part of cross-platform framework evaluations, and it is common to find comparative studies focusing on the differences between technical frameworks in terms of the number of programmatically accessible features [49, 59]. The importance of such access should be paramount to any app development approach, as a (cross-platform) app without access to device and platform features would be greatly limited in functionality compared to Native apps [61]. Access to these types of features is also commonly listed as a fundamental requirement in decision frameworks targeting the choice of development approach and technical frameworks, examples being that of Latif *et al.* 's study, stating that cross-platform frameworks must provide access to all of the features available [59, p. 4].

The bellow illustration shows of the connection between the cross-platform app, the intermediary abstraction layer providing programmatic access to device features, and some examples of such features available through the platform and device. What the intermediary layer is and how it works, depends on the development approach. Whereas the Hybrid, Progressive Web App and Interpreted approaches all will depend on such as an abstraction layer during runtime, the Cross-compiled and Model-Driven approaches typically do not, as the apps compiled using these approaches will have direct access to the underlying system and platform.

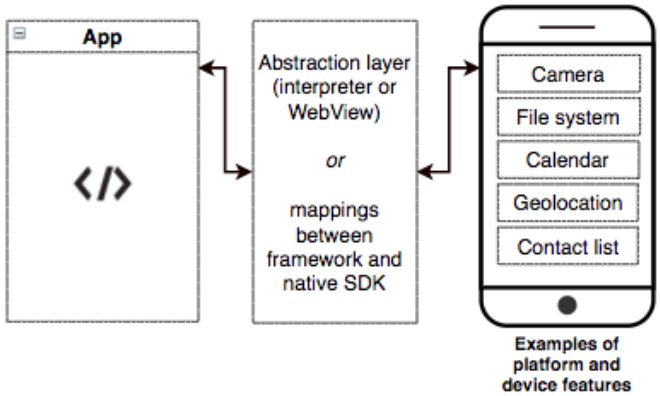

Fig. 6. Illustration of connection between app and device-platform features

In a newly published paper, we scrutinize the efficiency and ease of device feature access in Hybrid and Interpreted apps [11]. We found that both approaches do well at providing easy programmatic access to Native features, and we were unable to identify any features that could not be integrated and executed from within a cross-platform environment, wether the bridging technology was of the Interpreted or Hybrid/WebView-based approach. This paper is potentially a point of departure for proving or disconfirming previous claims on the subject, e.g. a study from Corral *et al.* discussing advantages and constraints of cross-platform development. In their 2012 paper, they state that cross-platform frameworks do not fully cater programmatic access to certain features [22, p. 1205], without emphasizing which features that are commonly missing.

A study by Palmieri *et al.* [72], also from 2012, compares five major cross-platform development frameworks of different approaches on a variety of parameters, including feature access. Their findings indicate that, while the statement from Corral [22] indeed to a degree reported correctly on the then status-quo of feature access, the level of access provided by the frameworks varied greatly. While the Hybrid approach, in the paper represented by PhoneGap, provided APIs to all but one of the compared features, the DragonRad framework supported only close to half of the

features listed. The same two frameworks are also scrutinized in a study by Ribeiro and da Silva [77]. However, in this study, the list of features used for comparison is less exhaustive than in [72], creating the illusion that DragonRAD can access the majority of such device features.

Drawing on the findings presented in newer research, thereof the previously mentioned study assessing feature access in the Hybrid and Interpreted approaches [11], the progress of cross-platform frameworks and technology can be noted with ease when reading papers of a certain age, especially those discussing platform and device feature access.

Another paper exemplifying progress within our field is a study by Smutný from 2012, discussing the benefits of *"going Native"* in the context of deciding on development approach. They claim that developing an app using the Native approach will allow for the best user experience if the app is to be used primarily offline [85, p. 654]. However, both cross-platform apps and Progressive Web Apps, as previously discussed, are offline-capable and functional, and should to a great extent not differ from the offline experience found in Native apps. The *Web* approach does, according to the authors of [85], not support offline mode, nevertheless this is no longer the case due to the possibilities of Progressive Web Apps.

The progress within device- and platform feature accessibility can be drawn from studies such as that by Escoffier and Lalanda [31], focusing on *heterogeneity and dynamism* in cross-platform apps. In regards to the status of feature access in 2015, the authors state that Cross-compiled solutions including Xamarin and Rhodes do not expose a feature-set vast enough to cater to the development of complex apps [31, p. 75]. These findings somewhat contradict those presented in Palmieri et al.'s [72] article, in which the authors state that Rhodes could in fact access the majority of the features implemented in their study. Also, the Xamarin website does not, as of December 2017, seem to agree with the claims presented by Escoffier and Lalanda [31]. In fact, this claim from 2015 might not any longer represent the state of the art, as the official statement from Xamarin fully contradicts it, claiming that Xamarin does in fact provide access to all the platform and device functionality available, and list features such as iBeacons and Android Fragments as examples of such [104].

Similar claims have also been identified pointing in the direction of the Web approach – thus indirectly Progressive Web Apps. Although regular Web apps and Progressive Web Apps both suffer from limitations imposed by web browsers, browser vendors and device feature availability exposed through JavaScript and HTML5 [13], we frequently encounter outdated claims related to which device features these development approaches have access to. Examples include programmatic access and control of device camera and GPS, which also in newer research (e.g. [58]) are reported inaccessible for these approaches. These claims contradict the findings provided by browser feature test platform Can I Use, stating that both these features can in fact be utilized in Web apps and Progressive Web Apps [15, 16] – the same is also reported by Ciman and Gaggi in their large-scale energy consumption experiment [19]. Nevertheless, other features are indeed not available from within a browser environment, examples being those of device calendar and contact list. In an attempt to better the situation for web apps, the WebAppBooster framework developed and presented by Puder *et al.* [76], is an independent service running in the background on a mobile device, listening to connections executed from localhost environments such as a browser. This allows websites running in a regular browser to call upon the WebAppBooster through the WebSockets protocol, which in turn executes some functionality only available through Native SDKs, including such as access to contact lists and sending of SMS. Not only is this approach novel and interesting, the article represent one of few scholarly research papers in which such a framework is presented outside of the Model-Driven Development approach, where academic tools seem more prevalent than those originating from practice.

5.1 Summary of section on Software Platform Features

What becomes evident is how newer papers may draw from- or rely too heavily on considerably older and thus potentially outdated research, and take the presented results for granted as to represent what is state of the art. One could argue that when newer papers base their technical claims on the findings presented in research of a certain age, it has the potential of creating a false impression of state of the art. In a fast-paced field such as mobile development, verifying such claims will be a continuous effort to stay relevant also for industry and practitioners.

Table 3. Summarizing research on Software Platform Features

Approach	Description	Frameworks encountered
Hybrid	An often-debated topic. Contrasted findings, whereas some report on feature availability by default in a set of frameworks, others extend the availability through plugins. A recent study report that Cordova-based frameworks do seemingly not pose limitations in feature access.	PhoneGap, Ionic
Interpreted	Difficult to conclude due to differences between frameworks. They vary in out-of-the-box feature exposure, but provide non-standardized plugin systems for bridging other features. Lack of standard results in plugin fragmentation.	MoSync, React Native
Cross-compiled	Academia contradict with practice. Studies report on major feature accessibility limitations, but e.g. Xamarin officially claim to provide access to the complete set of features. Fundamental differences between frameworks, concluding rendered difficult.	Xamarin, Rhodes, DragonRAD
Model-Driven	Limited by availability of pre-made models and Domain-Specific Language. Could in theory access all features due to transformation steps from models into Native code, but at least one study report of certain constraints.	MD^2, ADSML
Progressive Web Apps	Access to features do not depend on the technical framework. Rather, apps may only use of the features exposed through JavaScript and HTML5 APIs. Browsers vendors decide on API support, thus cross-browser compatibility is challenging.	-

6 PERFORMANCE AND HARDWARE UTILIZATION

The general perception in academia regarding performance of apps developed using cross-platform frameworks is that it is inherently subpar to Native apps due to abstraction layers such as interpreters, WebView engines and code transpilation- and compilation steps. An example of this perception is found in [53], a study by Huy and vanThanh on evaluation of development approaches (or *paradigms*), claiming that Hybrid apps are performance-wise inferior to Native apps due to the HTML rendering process that is required to take place in a WebView-based application, and that the end-result of this is that Hybrid apps cannot replace Native ones [53, p. 25]. Based on this statement, a discussion that could be had is whether or not Hybrid – or cross-platform approaches in general – are supposed to *replace* the Native approach, or merely provide alternative development techniques and possibilities in contexts benefiting from such.

Traversing the performance-oriented literature, several studies have been identified, most spanning a timeframe from 2012 through 2016. Of newer studies identified, we find that of Willocx *et al.* [102] from 2016, in which methods frequently found in performance studies are employed, including measuring memory consumption, disk space use and CPU usage on apps developed using a variety of cross-platform frameworks. They conducted a variety of tasks while recording device performance, one of which proved to be more efficient than its Native implementation baseline, which was that of page navigation in Hybrid apps. One of their most important contributions and findings indicate acceptable performance loss in cross-platform apps, although the penalty introduced heavily depend on approach and framework [102, p. 45]. Additionally, they report on differences between high-end and lower-end mobile devices, and that the penalty is specifically acceptable on the higher-end ones.

However, while some experiments, such as the aforementioned performance study find that cross-platform apps in general perform subpar to Native apps, one study identified report of the exact opposite. Ahti *et al.* [3] found that their Hybrid PhoneGap-based app had faster startup-time, consumed less memory and demanded much less disk space on Android compared to their Native app [3, p. 45]. This is in stark contrast to most literature, and the authors report of a PhoneGap-based implementation which had its weaknesses related to mostly user experience, but nevertheless provided a *"technically feasible alternative"* to Native development. Their reports of weaknesses in user experience and app user interface appearance could be due to their use of a traditional web interface library, Bootstrap, on top of PhoneGap. There are some Bootstrap alternatives they could have employed which expose HTML-based interface components mimicking both look and behaviour of those commonly found on each of the major platforms [47], two examples being Ionic Framework and Onsen UI. Nevertheless, their performance findings are of utmost interest, and contradict statements such as the aforementioned one by Huy and vanThanh [53, p. 25].

In addition to experiments such as those outlined above, we also identified a comprehensive study in the context of device energy consumption, by Ciman and Gaggi [19]. In their study, the authors measure the impact on energy consumption in an experiment including a wide array of device functionality and hardware sensors using four cross-platform frameworks – namely PhoneGap, Titanium and MoSync, and a regular web app. They find that it is of utmost importance to base the choice of development approach and framework on the software specification at hand, but that cross-platform apps developed will regardless introduce some form of an energy consumption penalty. The Interpreted approach is found to be more performant than they assumed, although an increase in device CPU usage was noted due to the need of runtime code interpretation. An interesting result from their experiment is that of the previous statement also hold true for apps of the Cross-compiled approach, which are in fact considered to be *"real Native application[s]"* [19, p. 16].

Drawing on these results, we find that the Cross-compiled based Xamarin framework, which should thus generate real Native apps, do in fact introduce a performance overhead when compared to the Native approach, according to Willocx *et al.* [101]. While their findings vary greatly between different devices and platforms, the Hybrid approach represented by PhoneGap tend to score inferior to both Xamarin and the Native approach, although there are test cases in which PhoneGap and Xamarin present close to identical results, being those of CPU usage and on-device installation size. Similar findings are also reported by Dhillon and Mahmoud [27] in their empirical investigation of technical frameworks, in which they find that the Cross-compiled MoSync framework perform with great differences on iOS and Android. While their iOS implementation performed well, the Android app performed subpar to what they expected from a framework of this approach.

Moreover, the well-cited study by Dalmasso *et al.* [24] in which they scrutinize and perform a comparison of cross-platform development frameworks, the authors reports on device performance using metrics including memory usage, power consumption and CPU usage. Because these metrics are frequently encountered in performance-oriented experiments, studying the progression of cross-platform performance through surveying existing literature could draw on papers such as this. The authors identify the Hybrid PhoneGap app to hardware-wise be more performant and efficient than the Interpreted Titanium app, although the trade-off is that of user experience and interface components which they found lacking in PhoneGap. While the experiment and the reported results contribute to the body of knowledge, a potential weakness in their research design is the lack of a Native app from which they could extract a performance baseline for the metrics measured. Such a baseline is present in similar studies, thereof e.g. [19, 23, 101]. Without a Native baseline, it would be inherently difficult to validate whether the performance measurements of the cross-platform apps rendered better or worse results than the performance goal – which should be *native-like performance*, according to several highly-cited studies [59, 105].

Although the Progressive Web Apps development approach is still in its infancy and thus have not yet seen much academic research effort, two studies were identified to scrutinize its performance. In [65], Malavolta *et al.* assess the possibility of an impact in energy consumption due to Service Workers executing and running background tasks. In brief, a Service Worker is a script that can run in the background as part of installed PWAs, conducting tasks such as data synchronization, push notifications and data caching [63]. While no significant impact was identified, the authors still suggest that inclusion of a Service Worker is a trade-off in terms of (minimal) energy consumption impact and the additional features it can provide. The latter is at the core of a study by Biørn-Hansen *et al.* [13], in which a feature comparison and performance measurement between the Native, Interpreted, Hybrid and PWA approaches are presented. The results implicate app launch- and interface render times on par with apps generated by the Hybrid and Interpreted approaches, while requiring multiple orders of magnitude less disk space.

The earliest performance study identified, an experiment by Corral *et al.* [23] from 2012 reveals that *native app performance* in cross-platform developed apps was not to be expected at the time of publishing, and reports an overall performance penalty in time lapsed for measured tasks (e.g. writing to and from disk file, requesting data from the network or GPS module). However, being that the penalty is only infrequently of any sizeable deviation from the Native baseline, the authors also state that the penalty should be fine for what they refer to as *"general-purpose business applications"* [23, p. 742], or line-of-business apps. A refresh and validation of this study would be of immense interest, although extended with more up-to-date frameworks and technologies for greater future validity and industry relevance. Their study only included a Native app and a Hybrid approach PhoneGap app, but the proliferation of Cross-compiled and Interpreted frameworks should in a newer study be accounted for. Also, a better understanding of differences in user expectation between business apps and regular apps could help in decision-making processes.

Within the Model-Driven (MDD) approach, studies targeting business contexts are frequently encountered (e.g. [30, 51, 64, 79]), and thus performance testing and surveying business user app expectations could be of importance to understand the target group. Alas, although there are numerous frameworks of the Model-Driven (MDD) approach originating from academia (e.g. [52, 54, 57]), we find that there is a general absence of performance discussion and empirical experiments examining the generated apps. An hypothesis for why this topic is so infrequently encountered in the literature could be that the models exposed by the frameworks translate to Native platform code, and should thus in theory not differ from Native apps [40]. However, in a study by Jia and Jones [54] on designing Domain-Specific Languages, the authors claim that also apps developed using MDD-based frameworks and tools suffer from performance penalties, something their own DSL, *ADSML*, is aiming to improve. Alas, no empirical evidence is provided to verify neither claim; that the average MDD tool suffer from performance constraints, nor that ADSML is *"[...] capable of generating high performance Native applications"* [54, p. 2]. Nevertheless, we find that suggestions for further research on frameworks within the MDD approach oftentimes include empirical verification of results (e.g. [94]). Reports of performance benchmarking of MDD generated apps are yet to be encountered in the literature identified, but could help to increase adaptation of such frameworks, as suggested in [64] by the developers of the MD^2 framework.

6.1 Summary of section on Performance and Hardware Utilization

Most performance-oriented papers seem to find cross-platform development a viable alternative to Native development regardless of the performance penalty introduced, although such a penalty differ greatly between the approaches and technical frameworks.

Table 4. Summarizing research on Performance and Hardware Utilization

Approach	Description	Frameworks encountered
Hybrid	Frequently debated, and in the literature reported to introduce performance penalties, but the level of severeness varies between studies. Acceptable performance is typically reported by experimental and empirical studies, while conceptual papers are more harsh in their review of the approach.	PhoneGap, Intel App Framework, Ionic, MGWT, jQuery Mobile, Sencha Touch, Famo.us
Interpreted	Reports of a performance penalty are frequently encountered due to the overhead of runtime interpretation of the code bases. For cross-platform apps where Native user interfaces are beneficial or required, the performance penalty can be perceived a trade-off.	Titanium, MoSync, Adobe AIR, React Native
Cross-compiled	Although apps generated using this approach are considered to be real Native apps, results of several performance experiments report of great discrepancy between the approaches. While some contexts might benefit from this approach, differences in performance between platforms and devices were considered significant.	Xamarin, MoSync
Model-Driven	No performance-oriented studies identified. Although the topic is frequently encountered in the literature as suggestions for further work, this has to the best of our knowledge not yet been followed up.	-
Progressive Web Apps	Energy consumption penalty caused by Service Workers reported insignificant although not necessarily negligible. Disk-space wise, orders of magnitude smaller than other cross-platform generated apps, with initial reporting of comparable performance.	-

7 SECURITY

Do cross-platform frameworks impose attack surfaces and security holes additional to those already exposed by the platforms and their SDKs? Not much research answering security-related questions has been identified, other than for Native (outside the scope for this survey) and the Hybrid development approach. Claims and statements without empirical grounding, as discussed in several of the previous sections, are also encountered in studies involving security. An example is that of a study by Lachgar *et al.* , reporting Hybrid apps' security to be *"not good"* [58, p. 112]. While lacking the empirical evidence or citations to back this claim, we find that questionnaires and interview-based based studies targeting developers seem to report of a similar notion: security is rarely a topic the participants touch upon (e.g. [2]).

The literature identified is primarily focused on the Android WebView, and the potential attack surfaces exposed by the component (e.g. [10, 48, 61, 106]). As previously discussed, the WebView is core to any Android or iOS Hybrid app [49], as it executes the web code (HTML, CSS and JavaScript) bundled together with a Native app wrapper. Most of the Hybrid approach security research identified is based on loading and executing malicious code, also known as Cross-Site Scripting (XSS), into the embedded WebView, which then can manipulate and execute Native code though the JavaScript bridges [10]. A 2017 study by Bao *et al.* [10] discusses an important point regarding loading of third-party libraries into a WebView codebase without actively traversing the codebases to check for malicious code and vulnerabilities. They display multiple ways to outright extract and thus steal data from a device through the Cordova interface exposed to the JavaScript environment, such as through vulnerable – yet work-as-intended – HTML and JavaScript snippets. Additionally, they stress that XSS attacks performed on mobile devices through Hybrid apps with deep platform integration can cause damage not typically found in website XSS attacks. In fact, extraction of device contact list, files and cookies are examples the authors list as what is possible to steal [10, p. 60].

This is also supported by Yu and Yamauchi's study [106] from 2013, proposing access control measures to better avoid execution of Native code outside of specific environments. Rather than working on securing the JavaScript codebase to any degree possible, the authors rather implement security measures at a Java object level, testing for threats when the JavaScript-to-native bridge (Foreign Function Interface) is registered. Thus, their proposed design allow users to be notified of possible threats, and enable them to manually accept or reject certain Java code from executing. This aligns with findings by Mercado *et al.* [68], reporting that security is of importance also to end-users, according to mining of app store reviews. The type of access control proposed by Yu and Yamauchi could also help developers identify third-party frameworks and libraries accessing device features in a malicious and harmful intent. For future research, a tool performing static analysis of Hybrid app codebases analyzing such malicious Native feature access could be of utmost interest for practitioners. Even though open source third-party libraries are indeed *open source*, reading through what could be thousands or tens of thousands of lines of code could be considered a bizarre activity for any commercial or hobbyist project.

Returning to the identified literature, another set of proposed guidelines to further secure apps running within WebView has been presented in a 2014 study by Hazarika *et al.* [48]. Emerging from their study is a requirement for Hybrid apps to only accept certain connections and implement policies for Native-device access,

> "This [native bridge/Foreign Function Interface] enables malicious JavaScript code to easily access critical information residing on the device. To handle such scenarios, the application must use secure connections with end to end encryption, as well follow a strict policy for file access." [48, p. 1592]

The core focus of most Hybrid approach security research has been the vulnerabilities of JavaScript and the possibilities of executing Native code through the WebView interfaces. This has also been noted in Chen *et al.* 's 2015 study [18] exploring HTML5-based injection attacks using HTML input text fields. While [10] also discuss HTML elements such as the image (`< img >`) tag, they have not discussed possible attack surfaces including input fields. The perceived novelty of Chen *et al.* 's experiment is grounded in the input field's ubiquitousness in mobile apps, although only four out of 8303 (0.048%) sampled apps proved to expose such a vulnerability. A comparison to other development approaches would be of utmost interest to better understand if this expose threats also found elsewhere, and proposed methods of alleviating the attack surface.

As becomes evident from the review, the majority of studies identified are targeted towards potential security gaps and *newly* introduced attack surfaces in Hybrid apps only. Alas, not much research on security has been found covering the other development approaches. Numerous papers discuss the importance of having a security perspective, but academia has to the best of our knowledge not followed up. This leaves an approachable gap for researchers to explore. An Hybrid app relies on the security measurements implemented by the Cordova framework for execution of code using a web browser's interpreter, however Interpreted apps execute on the device's language interpreter (V8 or JavaScriptCore) instead of inside the constraints of a browser [29]. Thus, apps of the Interpreted approach might enable for different attack surfaces than those found in Hybrid apps, and the same goes for the generative approaches including Cross-compiled and Model-Driven Development.

While studies on security outside the Hybrid approach is scarce, we do find mentions of it also in non-security oriented papers. In [27], Dhillon and Mahmoud evaluate, among a variety of parameters, the availability of two security measures in the Cross-compiled and Interpreted approaches, specifically code obfuscation possibilities and the frameworks' access to secure storage. Alas, their results illustrate a lack of implemented security in PhoneGap, while Titanium Appcelerator and Adobe Air varies in terms of obfuscation possibilities and secure storage access. The study does not provide much detail on either measure, but it is nevertheless a point of departure for further research to build on. The same is true for security studies on Model-Driven Development, where two papers were identified to merely briefly discuss the topic. In [97], Vaupel *et al.* mention how their MDD language map to platform-independent permission schemes, and how users are in control of which device features an app can take use of. The implementation of these measures were not discussed in any further depth. Security is also mentioned in a paper discussing one of the frequently encountered MDD frameworks, MD^2, in which the authors state that the framework does not implement any *"specific security features"* [64, p. 9] directly, although the provided DSL will inherently limit the possibilities for implementing malicious code. For future research, expanding the scope to look at Model-Driven Development not limited to the (cross-platform) mobile context might open for a broader perspective of what is available in terms of security, and possibly how security measures can be implemented in models and DSLs accordingly.

An additional factor at play when discussing security is the abnormally vast array of different Android devices, resulting in unique challenges for both Native and cross-platform development alike. In fact, one of the latest reports found focusing on the fragmented Android smartphone market is one by OpenSignal from 2015, in which they identify more than 24000 distinct Android-based devices [71] from a pool of 682000 devices having downloaded their app. Not only must developers deal with device fragmentation, also the version fragmentation is challenging according to the Android Developer platform [6]. The latest versions of the Android platform, Oreo (8.0 and 8.1) account for only 1.2% of the market, while Nougat (7.0 and 7.1) is installed on 28.5% of the devices surveyed. The older versions, including Marshmallow (6.0), Lollipop (5.0 and 5.1) and KitKat (4.4) still amount to 64.7% of the market. In their large-scale investigation on security in the fragmented

Android market, Mutchler *et al.* [70] uncover that version compatibility of apps, i.e. running apps targeting older SDKs on a new device, can in fact compromise certain security features added in later versions. Thus, future research on security in cross-platform Android apps should account for the fragmentation of the platform to ensure broad validity and generalizability of findings.

7.1 Summary of section on Security

We find that research gaps in the context of cross-platform security are both plentiful and of fundamental importance. Further research much-needed to better understand the potential attack surfaces available in cross-platform approaches and the end-result apps.

Table 5. Summarizing research on Security

Approach	Description	Frameworks encountered
Hybrid	Numerous papers addressing security issues, main focus on the Android WebView component and possibilities of Cross-Site Scripting attacks, injection of malicious code calling Native features for unlawful extraction of device data.	Cordova
Interpreted	We did not identify any studies assessing the security of Interpreted apps or frameworks, although inclusion of this approach in other studies on cross-platform development is common.	-
Cross-compiled	We did not identify any studies assessing the security of Cross-compiled apps or frameworks, although inclusion of this approach in other studies on cross-platform development is common.	-
Model-Driven	Only mentions of the importance of security, albeit no studies focusing on different aspects of security or attack surfaces introduced by frameworks or the approach itself were identified. Such studies could perhaps help the rate of adoption of MDD frameworks in the industry.	-
Progressive Web Apps	While there were no papers identified, but great research potential due to the approach's novelty, and possibilities of newly introduced attack surfaces.	-

8 TAXONOMY AND STATE OF RESEARCH

For each of the overarching concepts discussed throughout this review, we firstly comment on their state of research in relation to each of the overarching development approaches, then briefly comment on findings from sub-topics within each concept.

Table 6. An overview of taxonomy and state of research on cross-platform development

	Hybrid	Interpreted	Cross-compiled	Model-Driven	Progressive Web Apps
User Experience	Quantitative. Lacking qualitative.	Some qualitative. Insufficient.	Insufficient research.	Insufficient research.	No identified research.
Accessibility	*No identified research.*	*No identified research.*	*No identified research.*	*Insufficient research.*	*No identified research.*
Software Platform Features	Well-researched.	Well-researched.	Some research. Conclusion lacking.	Some research. Conclusion lacking.	Some research. Conclusion known.
Feature availability	*Debated. Likely not limited.*	*Debated. Likely not limited.*	*Debated. Likely not limited.*	*Likely not limited.*	*Browser constrained.*
Performance and Hardware	Well-researched. Dated.	Well-researched. Dated.	Well-researched. Dated.	Insufficient research.	Insufficient research.
Security	Concerning. Plentiful research.	Insufficient research.	Insufficient research.	No identified research.	No identified research.

In addition to the condensed and tabularized findings above, we also provide an overview of more technical nature, targeting the five development approaches discussed throughout this survey. The figure report on the current state of the art in terms of predominant programming languages, app execution environments, how user interfaces are rendered, and which controlling entity is managing access to device and platform features.

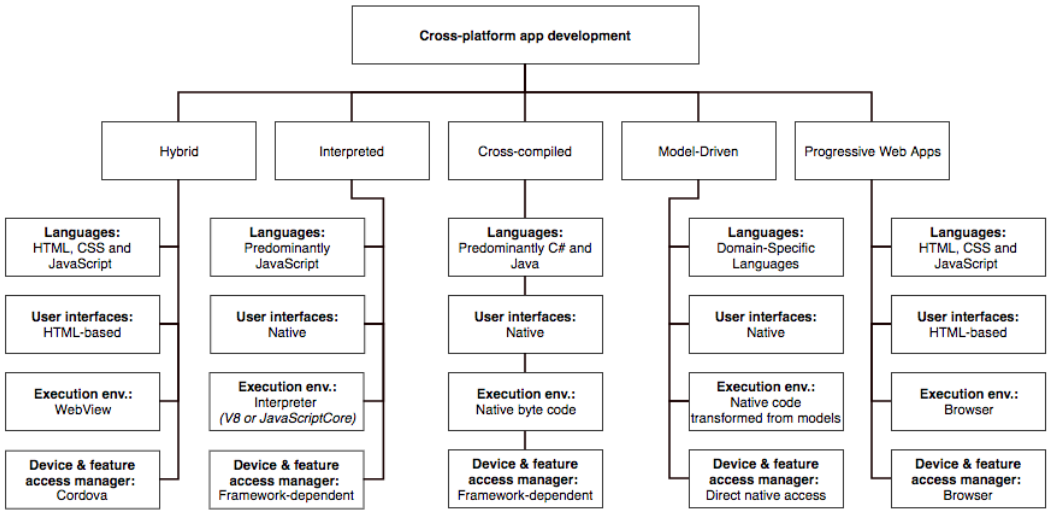

Fig. 7. Taxonomy of cross-platform development

9 CONCLUSION AND OUTLOOK

This section is divided in three, whereas we start by discussing a series of research challenges identified while analyzing the state of research. We then outline numerous suggestions for further work based on what we deem important and missing from the current literature, and discuss the evolution of cross-platform development while providing insight into a handful of interesting and novel projects and concepts. Lastly, we go on to conclude the study and reiterate on our contributions, as well as the problem areas identified and the motivation for conducting this research.

9.1 Research Challenges

The rapid pace of development conducted by practitioners and the proliferation of technical frameworks both impose core challenges for scholarly research, especially if it is to be of interest and use also for the industry. Research on cross-platform mobile development finds itself in a state in which newer research draw too heavily from the "state of the art" presented in older research, and present it in a fashion which makes it perceivable as still representing truth. A consequence of this is that, with the constant development of tools and frameworks, newer research may incorrectly present what the actual state of the art is. This is frequently encountered in the literature traversed and discussed in this review, where technically outdated or unbacked claims flourish. While lacking the rigor that is scientific research, practitioners are actively communicating everyday-findings through blogs, articles, newsletters, podcasts, conference articles and similar outlets. Due to the nature of rigorous academic research, non-academic contributions within our field should be considered of high importance and value for understanding the actual state of the art. Indeed, the pace at which practitioners can publish non-peer reviewed material cannot be beaten by peer-reviewed scientific work. Also, industry contributions are a great source researchers can tap into to find problem areas reported by practitioners.

The lack of qualitative studies involving users in research on user experience makes for a wide gap in the body of knowledge. In fact, only one qualitative study on user perception of cross-platform apps was identified [8], however this would need a technological refresh to represent the

current state of the art with regard to frameworks and approaches. This is also one of the most pressing challenges, as to validate the array of claims regarding user perception and experience using cross-platform apps. Interesting research challenges occur when basing experiments on human involvement, as to which research methods can be applied to verify the common claims identified. The qualitative study identified [8] conducted longitudinal and laboratory testing, but further research might also involve the use of advanced hardware, e.g. eye tracking equipment.

To rigorously scrutinize and report on the possibilities and constraints introduced by cross-platform development frameworks, the complexity of implementations and designs used for empirical evaluation should optimally be of that we commonly find in the app stores. Simple instantiations can wrongfully report admirable results within the constrains of a research project, but once the proposed guidelines/practices/concepts/technologies are applied to an industry or *real-life* setting, the research results may no longer be of the same validity. When developing new and novel instantiations for the sake of research, drawing from proposed requirements (e.g. [49, 58, 59]) and feature baselines (e.g. [12]) can perhaps be of help to map out what to measure and implement in order for the research to have maximum potential impact.

Keeping up with the fragmentation and ever-changing array of smartphone device types, screen resolutions, hardware and more impose serious research challenges more-so in the coming years with the advent of new smart devices (discussed in the next bullet point). Thus, a brief socio-economic discussion follows: what we typically find in academic research are technical labs powered by medium- to high-end devices, thus rendering the research inherently limited in validity and generality to countries in which such high-end devices are the norm, rather than lower-end ones.

9.2 Suggestions for Further Work

Several aspects of cross-platform development require further attention, as a lack of concluding evidence has been identified within all of the overarching concepts traversed in this survey. Heitkötter *et al.*'s seminal paper stress the importance of keeping up with the development and advancements of development frameworks and technologies along with the platforms they target [49, p. 309]. Based on their above suggestion, in an effort to provide a point of departure for those interested, we below discuss what we believe to be the imminent evolution of cross-platform mobile development, and thereafter suggest numerous ideas for future research to draw from.

9.2.1 Evolution of Cross-Platform Development. We have compiled a list of the most interesting projects we believe researchers and the industry will deal with in near future. Due to the nature of this field of practice and research, searching outside the academic sphere is an inherent requirement for researchers wanting to stay updated on technologies and the introduction of novel concepts.

Kotlin/Native by JetBrains enable use of the Java-based Kotlin programming language to run also on non-JVM (Java Virtual Machine) platforms, including e.g. the Apple iOS platform. Kotlin was introduced as an official programming language for the Android platform in 2017 [83], thus the Kotlin/Native framework may soon be used as a novel technology for Android developers to build cross-platform apps.

Flutter from Google provides a reactive framework for building cross-platform apps without the need of an interpreter or WebView, aiming at generating high-performing apps. Developers using Flutter will be writing in Dart, a Google-developed language with a syntax similar to Java/JavaScript. It uses Ahead-of-Time compilation to compile the Dart code into platform-specific Native code. Flutter implements their own set of interface widgets, allowing for the use of Google Material Design widgets on older Android devices [39].

Capacitor is the Ionic Framework's team novel alternative to Cordova [86]. The library will assist in the development of applications across mobile (Hybrid), mobile web (PWAs) and desktop

(Electron) through a set of standardized cross-context APIs. Due to the popularity of Cordova and the Hybrid approach, we hope research will scrutinize the possibilities and challenges introduced with Capacitor, especially so in relation to Cordova.

9.2.2 Suggestions Based on Surveyed Literature. Our suggestions below are those which we find typically recurring in the assessed literature, with emphasis on (i) those suggestions we have yet to find to have been targeted for (significant) research effort leading to any sorts of truth, and (ii) the suggestions we believe should receive continuous effort due to the nature of the field.

Continue researching user experience, user satisfaction and other Human-Computer Interaction aspects of cross-platform generated apps (e.g. [3, 14, 22, 23, 26, 41, 62]). Such studies could employ both quantitative and qualitative methods for data gathering to get a thorough understanding of user perception of cross-platform apps. This research could also help in answering hypotheses regarding developer bias, e.g. whether an app is only performant enough when accepted by an app developer, or if end-users have different thoughts and acceptance criteria regarding what a performant app is. While not explicitly listed by existing literature, we wish to encourage peers to focus on designing and experimenting with accessible cross-platform user interfaces.

Security (e.g. [10, 84, 106]) is alas an infrequently discussed topic in the literature traversed. Outside of security-oriented studies, only a few mentions of the issue have been recorded (e.g. [41, 80]). The lack of research on a topic of such importance leaves a fundamental gap in the body of knowledge, and should be targeted by researchers in future studies.

Performance and resource management (e.g. [20, 24, 26, 101, 105]) is an often-discussed topic in the literature, and have previously seen substantial research efforts. However, cross-platform mobile development being a fast-paced field of practice, continuous effort must be put forth to keep academia on track [49] with the industry, especially if novel concepts are to origin not only from the industry itself, but also from academia. This is applicable to areas such as technical- and evaluation frameworks, proposed standards, optimization patterns and more.

Compare free open-source software to closed and commercial solutions such as Xojo, Tabris.js. This entails evaluating differences in licensing, user interface components, Native plugins, community involvement, support and similar parameters [49], and could potentially be of paramount importance for the industry when deciding on cross-platform technology.

9.3 Conclusion

The landscape and ecosystem surrounding mobile application development involves a myriad of technological and conceptual options, a fact the four overarching development approaches and 74 different associated cross-platform frameworks listed in Table 1 are a testament to. Developers, designers and engineers face an overwhelming number of options to choose from and decisions to make, wherein there is no "one solution fits all" or even a silver bullet. Being a young field, cross-platform mobile development is likely to receive much more research effort going forward, as its application – while frequently criticized – is commonly reported as cost- and time-saving. While much of the current scientific body of knowledge is conceptual and descriptive of nature, more design-oriented studies including technical scrutiny of performance is needed to address the unbacked claims and statements all too common even in peer-reviewed published research. Together with the wide array of solutions and tools to choose from, the vast fragmentation of the mobile ecosystems [71] contributes to making the development of mobile apps particularly complex. Our contributions are that of a taxonomy, an overview of core concepts and domain terminology within the field of cross-platform app development, and a discussion on research challenges and thoughts on further work derived from our review. Further to this, to advance the field, we have

through the extensive overview in our contribution highlighted the grounding for and identified a number of research perspectives, angles and themes all worthy further pursue.

REFERENCES

[1] Timothy Yudi Adinugroho, Reina, and Josef Bernadi Gautama. 2015. Review of Multi-platform Mobile Application Development Using WebView: Learning Management System on Mobile Platform. In *Procedia Computer Science*, Vol. 59. Elsevier, 291–297. http://dx.doi.org/10.1016/j.procs.2015.07.568

[2] Arshad Ahmad, Kan Li, Chong Feng, Syed Mohammad Asim, Abdallah Yousif, and Shi Ge. 2018. An Empirical Study of Investigating Mobile Applications Development Challenges. *IEEE Access* 6 (2018), 17711–17728. http://dx.doi.org/10.1109/ACCESS.2018.2818724

[3] Ville Ahti, Sami Hyrynsalmi, and Olli Nevalainen. 2016. An Evaluation Framework for Cross-Platform Mobile App Development Tools: A Case Analysis of Adobe PhoneGap Framework. In *Proceedings of the 17th International Conference on Computer Systems and Technologies 2016 (CompSysTech '16)*. ACM, New York, NY, USA, 41–48. http://dx.doi.org/10.1145/2983468.2983484

[4] Ricardo Alcocer. 2013. But I Thought Titanium Was Cross Platform!? (23 July 2013). http://www.appcelerator.com/blog/2013/07/but-i-thought-titanium-was-cross-platform/ Accessed: 2017-8-3.

[5] Mohamed Ali and Ali Mesbah. 2016. Mining and Characterizing Hybrid Apps. In *Proceedings of the International Workshop on App Market Analytics (WAMA 2016)*. ACM, New York, NY, USA, 50–56. http://dx.doi.org/10.1145/2993259.2993263

[6] Android Developers. 2018. Dashboards. (Feb. 2018). https://developer.android.com/about/dashboards/index.html Accessed: 2018-2-12.

[7] Android Developers. N/A. WebView. (N/A). https://developer.android.com/reference/android/webkit/WebView.html Accessed: 2017-8-3.

[8] Esteban Angulo and Xavier Ferre. 2014. A Case Study on Cross-Platform Development Frameworks for Mobile Applications and UX. In *Proceedings of the XV International Conference on Human Computer Interaction*. ACM, 27. http://dl.acm.org/citation.cfm?id=2662253.2662280&coll=DL&dl=GUIDE

[9] Appcelerator. N/A. Appcelerator Platform. (N/A). http://docs.appcelerator.com/platform/latest/#!/guide/Titanium_Platform_Overview Accessed: 2018-2-1.

[10] Wenying Bao, Wenbin Yao, Ming Zong, and Dongbin Wang. 2017. Cross-site Scripting Attacks on Android Hybrid Applications. In *Proceedings of the 2017 International Conference on Cryptography, Security and Privacy*. ACM, 56–61. https://dl.acm.org/citation.cfm?id=3058076&CFID=819158537&CFTOKEN=38122944

[11] Andreas Biørn-Hansen and Gheorghita Ghinea. 2017. Bridging the Gap: Investigating Device-Feature Exposure in Cross-Platform Development. In *Proceedings of the 51st Hawaii International Conference on System Sciences*. ScholarSpace, 5717–5724.

[12] Andreas Biørn-Hansen, Tor-Morten Grønli, and Gheorghita Ghinea. 2017. Baseline Requirements for Comparative Research on Cross-Platform Mobile Development: A Literature Survey. In *Proceedings of the 30th Norwegian Informatics Conference*. Bibsys. http://ojs.bibsys.no/index.php/NIK/article/view/427

[13] Andreas Biørn-Hansen, Tim A Majchrzak, and Tor-Morten Grønli. 2018. Progressive Web Apps for the Unified Development of Mobile Applications. In *Web Information Systems and Technologies*, Tim A Majchrzak, Paolo Traverso, Karl-Heinz Krempels, and Valérie Monfort (Eds.). Lecture Notes in Business Information Processing, Vol. 322. Springer. http://dx.doi.org/10.1007/978-3-319-93527-0 To appear in LNBIP.

[14] Nader Boushehrinejadmoradi, Vinod Ganapathy, Santosh Nagarakatte, and Liviu Iftode. 2015. Testing Cross-Platform Mobile App Development Frameworks (T). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 441–451. http://dx.doi.org/10.1109/ASE.2015.21

[15] Can I Use. N/Aa. Geolocation API. (N/A). https://caniuse.com/#feat=geolocation Accessed: 2018-2-16.

[16] Can I Use. N/Ab. getUserMedia/Stream API. (N/A). https://caniuse.com/#feat=stream Accessed: 2018-2-16.

[17] Andre Charland and Brian LeRoux. 2011. Mobile Application Development: Web vs. Native. *Queueing Syst.* 9, 4 (1 April 2011), 20. http://dx.doi.org/10.1145/1966989.1968203

[18] Yen-Lin Chen, Hahn-Ming Lee, Albert B Jeng, and Te-En Wei. 2015. DroidCIA: A Novel Detection Method of Code Injection Attacks on HTML5-Based Mobile Apps. In *2015 IEEE Trustcom/BigDataSE/ISPA*, Vol. 1. IEEE, 1014–1021. http://dx.doi.org/10.1109/Trustcom.2015.477

[19] Matteo Ciman and Ombretta Gaggi. 2016. An empirical analysis of energy consumption of cross-platform frameworks for mobile development. *Pervasive and Mobile Computing* (26 Oct. 2016).

[20] Matteo Ciman, Ombretta Gaggi, and Nicola Gonzo. 2014. Cross-platform mobile development: a study on apps with animations. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. ACM, 757–759. http://dx.doi.org/10.1145/2554850.2555104

[21] Cordova. N/A. Architectural overview of Cordova platform. (N/A). https://cordova.apache.org/docs/en/latest/guide/overview/index.html Accessed: 2017-8-3.

[22] Luis Corral, Andrea Janes, and Tadas Remencius. 2012a. Potential Advantages and Disadvantages of Multiplatform Development Frameworks–A Vision on Mobile Environments. In *Procedia Computer Science*, Vol. 10. SciVerse ScienceDirect, 1202–1207. http://www.sciencedirect.com/science/article/pii/S1877050912005303

[23] Luis Corral, Alberto Sillitti, and Giancarlo Succi. 2012b. Mobile Multiplatform Development: An Experiment for Performance Analysis. *Procedia Comput. Sci.* 10 (Jan. 2012), 736–743. http://www.sciencedirect.com/science/article/pii/S1877050912004516

[24] Isabelle Dalmasso, Soumya Kanti Datta, Christian Bonnet, and Navid Nikaein. 2013. Survey, comparison and evaluation of cross platform mobile application development tools. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International*. IEEE, 323–328. http://dx.doi.org/10.1109/IWCMC.2013.6583580

[25] Lidija Davis. 2009. PhoneGap: People's Choice Winner at Web 2.0 Expo Launch Pad - ReadWrite. (2 April 2009). https://readwrite.com/2009/04/02/phone_gap_todays_peoples_choice_winner_at_launch_p/ Accessed: 2017-7-19.

[26] Lisandro Delia, Nicolas Galdamez, Pablo Thomas, Leonardo Corbalan, and Patricia Pesado. 2015. Multi-platform mobile application development analysis. In *2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS)*. 181–186. http://dx.doi.org/10.1109/RCIS.2015.7128878

[27] Sunny Dhillon and Qusay H Mahmoud. 2015. An evaluation framework for cross-platform mobile application development tools. *Softw. Pract. Exp.* 45, 10 (1 Oct. 2015), 1331–1357. http://dx.doi.org/10.1002/spe.2286

[28] W S El-Kassas, B A Abdullah, A H Yousef, and A Wahba. 2014. ICPMD: Integrated cross-platform mobile development solution. In *2014 9th International Conference on Computer Engineering Systems (ICCES)*. IEEE, 307–317. http://dx.doi.org/10.1109/ICCES.2014.7030977

[29] Wafaa S El-Kassas, Bassem A Abdullah, Ahmed H Yousef, and Ayman M Wahba. 2017. Taxonomy of Cross-Platform Mobile Applications Development Approaches. *Ain Shams Engineering Journal* 8, 2 (2017), 163–190. http://www.sciencedirect.com/science/article/pii/S2090447915001276

[30] Jan Ernsting, Christoph Rieger, Fabian Wrede, and Tim A Majchrzak. 2016. Refining a Reference Architecture for Model-Driven Business Apps. In *12th International Conference on Web Information Systems and Technologies*. Scitepress, 307–316. http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0005862103070316

[31] Clément Escoffier and Philippe Lalanda. 2015. Managing the Heterogeneity and Dynamism in Hybrid Mobile Applications. In *2015 IEEE International Conference on Services Computing*. IEEE, 74–81. http://dx.doi.org/10.1109/SCC.2015.20

[32] Clément Escoffier, Philippe Lalanda, and Ozan Gunalp. 2015. A Component Model to Manage the Heterogeneity and Dynamism in Mobile Applications. In *Proceedings of the 18th International ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE '15)*. ACM, New York, NY, USA, 85–90. http://dx.doi.org/10.1145/2737166.2737178

[33] El Hassane Ettifouri, Abdelkader Rhouati, Jamal Berrich, and Toumi Bouchentouf. 2017. Toward a Merged Approach for Cross-platform Applications (Web, Mobile and Desktop). In *Proceedings of the 2017 International Conference on Smart Digital Environment (ICSDE '17)*. ACM, New York, NY, USA, 207–213. http://dx.doi.org/10.1145/3128128.3128160

[34] European Commission. 2017. Web Accessibility. (9 May 2017). https://ec.europa.eu/digital-single-market/en/web-accessibility Accessed: 2017-8-15.

[35] Facebook. 2017. JavaScript Environment. (7 June 2017). https://facebook.github.io/react-native/docs/javascript-environment.html Accessed: 2017-8-3.

[36] Facebook. 2018. React Native. (2018). https://facebook.github.io/react-native/ Accessed: 2018-NA-NA.

[37] Maximiliano Firtman. 2016. *High Performance Mobile Web*. O'Reilly Media. http://shop.oreilly.com/product/0636920035060.do

[38] Maximiliano Firtman. 2018. Progressive Web Apps on iOS are here. https://medium.com/@firt/progressive-web-apps-on-ios-are-here-d00430dee3a7. (March 2018). https://medium.com/@firt/progressive-web-apps-on-ios-are-here-d00430dee3a7 Accessed: 2018-6-21.

[39] Flutter Team. 2017. FAQ - Flutter. (Dec. 2017). https://flutter.io/faq/ Accessed: 2017-12-14.

[40] Lamia Gaouar, Abdelkrim Benamar, and Fethi Tarik Bendimerad. 2015. Model Driven Approaches to Cross Platform Mobile Development. In *Proceedings of the International Conference on Intelligent Information Processing, Security and Advanced Communication (IPAC '15)*. ACM, New York, NY, USA, 19:1–19:5. http://dx.doi.org/10.1145/2816839.2816882

[41] Lamia Gaouar, Abdelkrim Benamar, and Fethi Tarik Bendimerad. 2016. Desirable Requirements of Cross Platform Mobile Development Tools. *Electronic Devices* 5 (March 2016), 14–22. http://www.dline.info/ed/fulltext/v5n1/edv5n1_3.pdf

[42] Matt Gaunt. 2018. Service Workers: an Introduction. (Jan. 2018). https://developers.google.com/web/fundamentals/primers/service-workers/ Accessed: 2018-1-22.

[43] Nizamettin Gok and Nitin Khanna. 2013. *Building Hybrid Android Apps with Java and JavaScript*. O'Reilly Media, Incorporated. http://shop.oreilly.com/product/0636920028994.do

[44] Tony Gorschek, Ewan Tempero, and Lefteris Angelis. 2014. On the use of software design models in software

development practice: An empirical investigation. *J. Syst. Softw.* 95 (Sept. 2014), 176–193. http://dx.doi.org/10.1016/j.jss.2014.03.082

[45] Chris Griffith. 2017. *Mobile App Development with Ionic2: Cross-Platform Apps with Ionic 2, Angular 2, and Cordova*. O'Reilly Media. http://shop.oreilly.com/product/0636920044710.do

[46] Tor-Morten Grønli and Gheorghita Ghinea. 2016. Meeting Quality Standards for Mobile Application Development in Businesses: A Framework for Cross-Platform Testing. In *2016 49th Hawaii International Conference on System Sciences (HICSS)*. 5711–5720. http://dx.doi.org/10.1109/HICSS.2016.706

[47] Tor-Morten Gronli, Jarle Hansen, Gheorghita Ghinea, and Muhammad Younas. 2014. Mobile Application Platform Heterogeneity: Android vs Windows Phone vs iOS vs Firefox OS. In *2014 IEEE 28th International Conference on Advanced Information Networking and Applications*. IEEE, 635–641. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6838724

[48] Pinku Hazarika, Rahul Raj CP, and Seshubabu Tolety. 2014. Recommendations for Webview Based Mobile Applications on Android. In *2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies*. IEEE, 1589–1592. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7019375

[49] Henning Heitkötter, Sebastian Hanschke, and Tim A Majchrzak. 2012a. Comparing Cross-platform Development Approaches for Mobile Applications. In *Proceedings 8th WEBIST*. SciTePress, 299–311.

[50] Henning Heitkötter, Sebastian Hanschke, and Tim A Majchrzak. 2012b. Evaluating Cross-Platform Development Approaches for Mobile Applications. In *Web Information Systems and Technologies*. Springer Berlin Heidelberg, 120–138. http://link.springer.com/chapter/10.1007/978-3-642-36608-6_8

[51] Henning Heitkötter and Tim A Majchrzak. 2013. Cross-Platform Development of Business Apps with MD2. In *Design Science at the Intersection of Physical and Virtual Design (Lecture Notes in Computer Science)*. Springer, Berlin, Heidelberg, 405–411. http://dx.doi.org/10.1007/978-3-642-38827-9_29

[52] Henning Heitkötter, Tim A Majchrzak, and Herbert Kuchen. 2013. Cross-platform Model-driven Development of Mobile Applications with Md2. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing (SAC '13)*. ACM, New York, NY, USA, 526–533. http://dl.acm.org/citation.cfm?id=2480464

[53] Ngu Phuc Huy and Do vanThanh. 2012. Evaluation of mobile app paradigms. In *Proceedings of the 10th International Conference on Advances in Mobile Computing & Multimedia*. ACM, 25–30. http://dx.doi.org/10.1145/2428955.2428968

[54] Xiaoping Jia and Christopher Jones. 2015. Design of adaptive domain-specific modeling languages for model-driven mobile application development. In *2015 10th International Joint Conference on Software Technologies (ICSOFT)*, Vol. 1. IEEE, 1–6. http://ieeexplore.ieee.org/abstract/document/7521157/

[55] Mona Erfani Joorabchi, Ali Mesbah, and Philippe Kruchten. 2013. Real Challenges in Mobile App Development. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*. CPS, 15–24. http://dx.doi.org/10.1109/ESEM.2013.9

[56] Elmar Krainz, Johannes Feiner, and Martin Fruhmann. 2016. Accelerated Development for Accessible Apps – Model Driven Development of Transportation Apps for Visually Impaired People. In *Human-Centered and Error-Resilient Systems Development*. Springer, Cham, 374–381. http://dx.doi.org/10.1007/978-3-319-44902-9

[57] Dean Kramer, Tony Clark, and Samia Oussena. 2010. MobDSL: A Domain Specific Language for multiple mobile platform deployment. In *2010 IEEE International Conference on Networked Embedded Systems for Enterprise Applications*. IEEE, 1–7. http://dx.doi.org/10.1109/NESEA.2010.5678062

[58] Mohamed Lachgar and Abdelmounaïm Abdali. 2017. Decision Framework for Mobile Development Methods. *International Journal of Advanced Computer Science and Applications* 8, 2 (2017), 110–118. http://thesai.org/Publications/ViewPaper?Volume=8&Issue=2&Code=ijacsa&SerialNo=15

[59] Mounaim Latif, Younes Lakhrissi, El Habib Nfaoui, and Najia Es-Sbai. 2016. Cross platform approach for mobile application development: A survey. In *2016 International Conference on Information Technology for Organizations Development (IT4OD)*. IEEE, 1–5. http://dx.doi.org/10.1109/IT4OD.2016.7479278

[60] Olivier Le Goaer and Sacha Waltham. 2013. Yet Another DSL for Cross-platforms Mobile Development. In *Proceedings of the First Workshop on the Globalization of Domain Specific Languages (GlobalDSL '13)*. ACM, New York, NY, USA, 28–33. http://doi.acm.org/10.1145/2489812.2489819

[61] Tongbo Luo, Hao Hao, Wenliang Du, Yifei Wang, and Heng Yin. 2011. Attacks on WebView in the Android system. In *Proceedings of the 27th Annual Computer Security Applications Conference*. ACM, 343–352. http://dx.doi.org/10.1145/2076732.2076781

[62] Tim Majchrzak, Andreas Biørn-Hansen, and Tor-Morten Grønli. 2017. Comprehensive Analysis of Innovative Cross-Platform App Development Frameworks. In *Proceedings of the 50th Hawaii International Conference on System Sciences*. scholarspace.manoa.hawaii.edu, 6162–6171. http://hdl.handle.net/10125/41909

[63] Tim A Majchrzak, Andreas Biørn-Hansen, and Tor-Morten Grønli. 2018. Progressive Web Apps: the Definite Approach to Cross-Platform Development?. In *Proceedings of the 51st Hawaii International Conference on System Sciences*. ScholarSpace, 5735–5745. http://hdl.handle.net/10125/50607

[64] Tim A Majchrzak and Jan Ernsting. 2015. Achieving business practicability of model-driven cross-platform apps. *Open Journal of Information Systems* 2, 2 (2015), 3–14. http://hdl.handle.net/11250/2392249

[65] Ivano Malavolta, Giuseppe Procaccianti, Paul Noorland, and Petar Vukmirović. 2017. Assessing the Impact of Service Workers on the Energy Efficiency of Progressive Web Apps. In *Proceedings of the 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft '17)*. IEEE Press, Piscataway, NJ, USA, 35–45. http://dl.acm.org/citation.cfm?id=3104093

[66] Ivano Malavolta, Stefano Ruberto, Tommaso Soru, and Valerio Terragni. 2015a. End Users' Perception of Hybrid Mobile Apps in the Google Play Store. In *2015 IEEE International Conference on Mobile Services*. IEEE, 25–32. http://ieeexplore.ieee.org/document/7226668/?reload=true&arnumber=7226668

[67] Ivano Malavolta, Stefano Ruberto, Tommaso Soru, and Valerio Terragni. 2015b. Hybrid Mobile Apps in the Google Play Store: An Exploratory Investigation. In *Proceedings of the Second ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft '15)*. IEEE Press, Piscataway, NJ, USA, 56–59. http://dl.acm.org/citation.cfm?id=2825051

[68] Iván Tactuk Mercado, Nuthan Munaiah, and Andrew Meneely. 2016. The Impact of Cross-platform Development Approaches for Mobile Applications from the User's Perspective. In *Proceedings of the International Workshop on App Market Analytics (WAMA 2016)*. ACM, New York, NY, USA, 43–49. http://dl.acm.org/citation.cfm?id=2825041.2825051

[69] MoSync AB. 2015. MoSync. (2015). https://github.com/MoSync/MoSync

[70] Patrick Mutchler, Yeganeh Safaei, Adam Doupé, and John Mitchell. 2016. Target Fragmentation in Android Apps. In *2016 IEEE Security and Privacy Workshops (SPW)*. IEEE, 204–213. http://dx.doi.org/10.1109/SPW.2016.31

[71] OpenSignal. 2015. *Android Fragmentation Visualized*. Technical Report. https://opensignal.com/legacy-assets/pdf/reports/2015_08_fragmentation_report.pdf

[72] Manuel Palmieri, Inderjeet Singh, and Antonio Cicchetti. 2012. Comparison of cross-platform mobile development tools. In *2012 16th International Conference on Intelligence in Next Generation Networks*. IEEE, 179–186. http://dx.doi.org/10.1109/ICIN.2012.6376023

[73] Kim Patch, Jeanne Spellman, and Kathy Wahlbin. 2015. Mobile Accessibility: How WCAG 2.0 and Other W3C/WAI Guidelines Apply to Mobile. (26 Feb. 2015). https://www.w3.org/TR/mobile-accessibility-mapping/ Accessed: 2017-8-15.

[74] Joachim Perchat, Mikael Desertot, and Sylvain Lecomte. 2013. Component based Framework to Create Mobile Cross-platform Applications. In *Procedia Computer Science*, Vol. 19. ScienceDirect, 1004–1011. http://www.sciencedirect.com/science/article/pii/S1877050913007485

[75] Marc Pous, Circe Serra-Vallmitjana, Rafael Giménez, Marc Torrent-Moreno, and David Boix. 2012. Enhancing accessibility: Mobile to ATM case study. In *Consumer Communications and Networking Conference (CCNC), 2012 IEEE*. IEEE, 404–408. http://dx.doi.org/10.1109/CCNC.2012.6181024

[76] Arno Puder, Nikolai Tillmann, and Michał Moskal. 2014. Exposing native device APIs to web apps. In *Proceedings of the 1st International Conference on Mobile Software Engineering and Systems*. ACM, 18–26. http://dx.doi.org/10.1145/2593902.2593908

[77] André Ribeiro and Alberto Rodrigues da Silva. 2012. Survey on Cross-Platforms and Languages for Mobile Apps. In *2012 Eighth International Conference on the Quality of Information and Communications Technology*. IEEE, 255–260. http://dx.doi.org/10.1109/QUATIC.2012.56

[78] António Nestor Ribeiro and Costa Rogério Araújo. 2016. An Automated Model Based Approach to Mobile UI Specification and Development. In *Human-Computer Interaction. Theory, Design, Development and Practice (Lecture Notes in Computer Science)*. Springer, Cham, 523–534. http://dx.doi.org/10.1007/978-3-319-39510-4

[79] Christoph Rieger. 2018. Evaluating a Graphical Model-Driven Approach to Codeless Business App Development. In *Proceedings of the 51st Hawaii International Conference on System Sciences*. ScholarSpace, 5725–5735. http://hdl.handle.net/10125/50606

[80] Christoph Rieger and Tim A Majchrzak. 2016. Weighted Evaluation Framework for Cross-Platform App Development Approaches. In *Information Systems: Development, Research, Applications, Education (Lecture Notes in Business Information Processing)*, Stanislaw Wrycza (Ed.). Springer International Publishing, 18–39. http://link.springer.com/chapter/10.1007/978-3-319-46642-2_2

[81] Christoph Rieger and Tim A Majchrzak. 2018. A Taxonomy for App-Enabled Devices: Mastering the Mobile Device Jungle. In *Lecture Notes in Business Information Processing: Web Information Systems and Technologies*. Vol. 322. Springer International Publishing, 202–220. http://dx.doi.org/10.1007/978-3-319-93527-0_10

[82] Dagfinn Rømen and Dag Svanæs. 2012. Validating WCAG versions 1.0 and 2.0 through usability testing with disabled users. *Univ Access Inf Soc* 11, 4 (1 Nov. 2012), 375–385. http://dx.doi.org/10.1007/s10209-011-0259-3

[83] Maxim Shafirov. 2017. Kotlin on Android. Now official. (17 May 2017). https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/ Accessed: 2017-6-9.

[84] Mohamed Shehab and Abeer AlJarrah. 2014. Reducing Attack Surface on Cordova-based Hybrid Mobile Apps. In *Proceedings of the 2Nd International Workshop on Mobile Development Lifecycle*. ACM, New York, NY, USA, 1–8.

http://dx.doi.org/10.1145/2688412.2688417
[85] Pavel Smutný. 2012. Mobile development tools and cross-platform solutions. In *Proceedings of the 13th International Carpathian Control Conference (ICCC)*. IEEE, 653–656. http://dx.doi.org/10.1109/CarpathianCC.2012.6228727
[86] Ben Sperry, Max Lynch, Adam Bradley, and Justin Willis. 2017. Ionic Show. (Dec. 2017). https://www.youtube.com/watch?v=YhlRbFQotT4
[87] Statista. 2017a. Annual number of mobile app downloads worldwide 2021 | Statistic. (2017). https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/ Accessed: 2017-7-18.
[88] Statista. 2017b. App stores: number of apps in leading app stores 2017 | Statista. (2017). https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/ Accessed: 2017-7-18.
[89] Statista. 2017c. Apple App Store: number of available apps 2017 | Statistic. (2017). https://www.statista.com/statistics/263795/number-of-available-apps-in-the-apple-app-store/ Accessed: 2017-7-18.
[90] Statista. 2017d. Mobile app revenues 2015-2020 | Statistic. (2017). https://www.statista.com/statistics/269025/worldwide-mobile-app-revenue-forecast/ Accessed: 2017-7-18.
[91] Statista. 2017e. Number of Google Play Store apps 2017 | Statistic. (2017). https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/ Accessed: 2017-7-18.
[92] Telerik. N/A. NativeScript. https://www.nativescript.org/. (N/A). https://www.nativescript.org/ Accessed: 2015-10-21.
[93] Eric Umuhoza and Marco Brambilla. 2016. Model Driven Development Approaches for Mobile Applications: A Survey. In *Mobile Web and Intelligent Information Systems (Lecture Notes in Computer Science)*. Springer, Cham, 93–107. https://link.springer.com/chapter/10.1007/978-3-319-44215-0_8
[94] Eric Umuhoza, Hamza Ed-douibi, Marco Brambilla, Jordi Cabot, and Aldo Bongio. 2015. Automatic Code Generation for Cross-platform, Multi-device Mobile Apps: Some Reflections from an Industrial Experience. In *Proceedings of the 3rd International Workshop on Mobile Development Lifecycle (MobileDeLi 2015)*. ACM, New York, NY, USA, 37–44. http://dx.doi.org/10.1145/2846661.2846666
[95] Usability.gov. 2013. System Usability Scale (SUS). (6 Sept. 2013). https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html Accessed: 2017-8-17.
[96] Muhammad Usman, Muhammad Zohaib Iqbal, and Muhammad Uzair Khan. 2017. A product-line model-driven engineering approach for generating feature-based mobile applications. *Journal of Systems and Software* 123 (Jan. 2017), 1–32. http://dx.doi.org/10.1016/j.jss.2016.09.049
[97] Steffen Vaupel, Gabriele Taentzer, Jan Peer Harries, Raphael Stroh, René Gerlach, and Michael Guckert. 2014. Model-Driven Development of Mobile Applications Allowing Role-Driven Variants. In *Model-Driven Engineering Languages and Systems (Lecture Notes in Computer Science)*. Springer, Cham, 1–17. http://dx.doi.org/10.1007/978-3-319-11653-2_1
[98] Nicolas Viennot, Edward Garcia, and Jason Nieh. 2014. A Measurement Study of Google Play. In *The 2014 ACM international conference on Measurement and modeling of computer systems*, Vol. 42. ACM, New York, NY, USA, 221–233. http://dx.doi.org/10.1145/2637364.2592003
[99] W3C. 2017. Web Accessibility Laws and Policies. (19 July 2017). https://www.w3.org/WAI/Policy/ Accessed: 2017-8-15.
[100] Web Accessibility Initiative. 2016. Mobile Accessibility. (Aug. 2016). https://www.w3.org/WAI/mobile/ Accessed: 2017-10-20.
[101] Michiel Willocx, Jan Vossaert, and Vincent Naessens. 2015. A Quantitative Assessment of Performance in Mobile App Development Tools. In *Mobile Services (MS), 2015 IEEE International Conference on*. IEEE, 454–461. http://dx.doi.org/10.1109/MobServ.2015.68
[102] Michiel Willocx, Jan Vossaert, and Vincent Naessens. 2016. Comparing Performance Parameters of Mobile App Development Strategies. In *2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 38–47. http://dx.doi.org/10.1109/MobileSoft.2016.028
[103] Xamarin. 2017. How Does Xamarin Work? (March 2017). https://developer.xamarin.com/guides/cross-platform/getting_started/introduction_to_mobile_development/#How_Does_Xamarin_Work Accessed: 2018-2-13.
[104] Xamarin. N/A. Mobile Application Development to Build Apps in C#. (N/A). https://www.xamarin.com/platform Accessed: 2017-10-20.
[105] Spyros Xanthopoulos and Stelios Xinogalos. 2013. A Comparative Analysis of Cross-platform Development Approaches for Mobile Applications. In *Proceedings of the 6th Balkan Conference in Informatics (BCI '13)*. ACM, 213–220. http://doi.acm.org/10.1145/2490257.2490292
[106] Jing Yu and Toshihiro Yamauchi. 2013. Access Control to Prevent Attacks Exploiting Vulnerabilities of WebView in Android OS. In *2013 IEEE 10th International Conference on High Performance Computing and Communications 2013 IEEE International Conference on Embedded and Ubiquitous Computing*. 1628–1633. http://dx.doi.org/10.1109/HPCC.and.EUC.2013.229

June 2018Month Year

Received February 2018; revised July 2018; accepted August 0000; revised February 2018