The Journal of Systems and Software 153 (2019) 175–199

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Towards the definitive evaluation framework for cross-platform app development approaches

Christoph Rieger[a], Tim A. Majchrzak[b,*]

[a] ERCIS, University of Münster, Münster, Germany
[b] ERCIS, University of Agder, Kristiansand, Norway

ARTICLE INFO

Article history:
Received 20 November 2018
Revised 25 February 2019
Accepted 1 April 2019
Available online 2 April 2019

Keywords:
Mobile app
Mobile computing
Cross-platform
Multi-platform
Development framework

ABSTRACT

Mobile app development is hindered by device fragmentation and vendor-specific modifications. Boundaries between devices blur with PC-tablet hybrids on the one side and wearables on the other. Future apps need to support a host of app-enabled devices with differing capabilities, along with their software ecosystems. Prior work on cross-platform app development concerned concepts and prototypes, and compared approaches that target smartphones. To aid choosing an appropriate framework and to support the scientific assessment of approaches, an up-to-date comparison framework is needed. Extending work on a holistic, weighted set of assessment criteria, we propose what could become the definitive framework for evaluating cross-platform approaches. We have based it on sound abstract concepts that allow extensions. The weighting capabilities offer customisation to avoid the proverbial comparison of apples and oranges lurking in the variety of available frameworks. Moreover, it advises on multiple development situations based on a single assessment. In this article, we motivate and describe our evaluation criteria. We then present a study that assesses several frameworks and compares them to Web Apps and native development. Our findings suggest that cross-platform development has seen much progress but the challenges are ever growing. Therefore, additional support for app developers is warranted.

1. Introduction

With iOS and Android, only two platforms with significant market share remain for the development of mobile apps (Forni and van der Meulen, 2017). Nevertheless, cross-platform technology continues to be very important and many app development frameworks exist (Heitkötter et al., 2013a; El-Kassas et al., 2017). When creating apps, there is still no uniform recommendation whether – or in which case – to employ web technology, a cross-platform approach, or a native Software Development Kit (SDK) (Rieger and Majchrzak, 2016). The emergence of Progressive Web Apps (PWA) has on the one hand brought a contestant for unification (Majchrzak et al., 2018; Biørn-Hansen et al., 2017); on the other hand, it underlines that professional developers still seek for *easier* ways of developing once but having their app run on multiple platforms. There seems to be profound interest in straightforward yet customisable solutions, for instance demonstrated in trivial patents (such as for "customizing a mobile application using a web-based interface" (Brisebois et al., 2017)).

The complexity of app development does not merely come from the need to cover two more or less incompatible platforms. Device fragmentation and vendor-specific modifications incur that particularly developing for Android is not uniform (Dobie, 2012). Additionally, the boundaries between devices are blurring with PC-tablet hybrids or wearables which extend computing into domains of watches, formerly unconnected electronic helpers, and even clothing (Nanjappan et al., 2017). There is a *jungle* of *app-enabled* devices (Rieger and Majchrzak, 2018), each posing specific capabilities and idiosyncrasies. The different categories of devices also bring their own ecosystems and contexts of usage. It is easy to imagine that developing an app supposed to run on a smartphone as well as within the system of a car (Wolf, 2013) and additionally on a screenless smart home/Internet-of-Things (IoT) device (Alaa et al., 2017) poses a tremendous challenge. This convergence of smart, user-targeted gadgets and formerly hidden small-scale information technology will need to be reflected in the development approaches used in the future.

Prior work on cross-platform app development has mainly concerned been with two topics. First, concepts and prototypes were

* Corresponding author.
E-mail addresses: christoph.rieger@ercis.de (C. Rieger), tima@ercis.de (T.A. Majchrzak).

https://doi.org/10.1016/j.jss.2019.04.001

proposed, such as by Heitkötter and Majchrzak (2013) and like applause (Friese, 2014). Second, frameworks that target smartphones and tablets were compared, for example by Heitkötter et al. (2013a), Ohrt and Turau (2012), and Le Goaer and Waltham (2013). To aid developers in choosing an appropriate framework and to support the scientific assessment of approaches, an update to the comparison frameworks is needed. It ought to consider the recent technological developments both regarding the capabilities of cross-platform frameworks and novel ideas such as PWAs. At the same time, it needs to reflect the demands from developers while not compromising academic rigour. Moreover, it needs to take into account that the consequences from using cross-platform technology impact the user experience (UX). Therefore, wisely using such approaches is indicated (Mercado et al., 2016), and this line of thinking should be reflected in any attempt to evaluate approaches.

Extending the first attempt to provide a holistic, weighted set of assessment criteria by Rieger and Majchrzak (2016), we propose steps towards what we believe could become the *definite* framework for evaluating cross-platform development approaches. We have based it on sound abstract concepts that allow adaptability to future technological developments. It seeks to be versatile and handy for practitioners yet fulfil what is needed to satisfy a critical scientific assessment. The weighting capabilities offer individualisation and customisation. We, thus, ensure that the proverbial comparison of apples to oranges that lurks in the variety of available frameworks and goals of development is avoided. Moreover, our evaluation framework offers the opportunity to get advice for multiple development endeavours based on a single assessment.

Please note the usage of terms in the following. Framework is *overloaded* as it refers to our evaluation framework as well as to software frameworks for cross-platform app development. Unless we refer to the evaluation, framework denotes the latter. If the context poses the chance of confusion, we qualify framework with "evaluation". We speak of a cross-platform development approach when we do not consider a concrete software implementation but rather the general way of solving the challenge of developing one but running apps on several platforms.

This article makes a number of contributions. First, it provides an evaluation framework for cross-platform development approaches for app-enabled devices. It can not only be used as provided but the extensive criteria catalogue serves as a reference which may also be employed for purposes beyond our framework. Second, we provide weight profiles to be used in conjunction with the framework. These profiles enable a non-generic usage of the framework, which allows users to adapt it to their company- or project-specific needs. Third, we present the results from an exemplary study with several development approaches, including (Progressive) Web Apps, hybrid apps, runtime approaches, and native development for comparison. The study not only seeks to underline the feasibility of our framework but to provide concrete advice.

The remainder of this article is structured as follows. In Section 2 we give an overview of works that provide a precondition to ours, are complementary, or are otherwise related in content or approach. Section 3 comprehensively presents our criteria catalogue and the rationale behind each of the criteria. It thereby serves both as a core scientific contribution and as a reference. The evaluation criteria are motivated and explained in detail; where applicable, examples are given for better illustration. Section 4 proposes weight profiles, which can be used to evaluate development approaches in a customized fashion. To demonstrate the feasibility of our work and to give practical recommendations, we present an evaluation study in Section 5. Our findings are then discussed in Section 6, which includes results from expert feedback, a proposal for a research agenda, limitations, and indication of our future work. Finally, in Section 7 we draw a conclusion.

2. Related work

The work presented in this article draws from the field of cross-platform development frameworks, which has emerged since the advent of smartphones. In addition, it takes account of the recent developments in the domain of mobile devices and the implications on future mobile app development approaches. Therefore, related work on both fields is presented in the following.

2.1. Cross-platform frameworks

Resulting from the increasing popularity of cross-platform development frameworks, a multitude of scientific works has been prepared for this topic. However, most papers are of experimental nature and restricted to single frameworks, or limited by the choice of considered development approaches. Only few provide a thorough evaluation based on a diverse set of criteria. A comprehensive summary of related literature regarding covered tools, criteria[1], and focal areas of comparison is given in Table 1. It combines a literature search within the scientific database Scopus on evaluations of mobile cross-platform frameworks, using the query

```
TITLE-ABS-KEY((comparison OR evaluation OR
review OR survey) AND
  (mobile OR app OR wearable OR application
OR vehicular OR ‘‘in-vehicle’’) AND
  (’’cross-platform’’ OR ‘‘multi-platform’’
OR ‘‘cross platform’’) AND
  (framework OR approach))
```

combined with a forward search on the papers by Heitkötter et al. (2012) and Heitkötter et al. (2013a). The latter represent early work on systematic assessment of app development frameworks for smartphones and have been used by many authors as basis for further research on apps. Examples include the definition of quality criteria for HTML5 frameworks (Sohn et al., 2015), quantitative performance evaluations (Willocx et al., 2015), and the creation of cross-platform development frameworks such as ICPMD (El-Kassas et al., 2014) and MD^2 (Heitkötter et al., 2015). To put the identified literature into context, we highlight notable details in the following.

Early papers have typically only considered few criteria – if at all (Rahul Raj and Tolety, 2012; Sansour et al., 2014). It can be noticed that few works perform a rather comprehensive evaluation, often neglecting a user's perspective on cross-platform app development (cf., e.g., El-Kassas et al., 2017). For example, Ohrt and Turau (2012) have analysed nine tools with regard to developers' needs and user expectations. Many papers focus on particular aspects of apps such as animations (Ciman et al., 2014), performance (Dalmasso et al., 2013), or energy consumption (Ciman and Gaggi, 2015).

We can also observe that the set of considered criteria does not appear to be coherent over time. Criteria are often grouped into common categories (Ohrt and Turau, 2012; Xanthopoulos and Xinogalos, 2013; Hudli et al., 2015; Sommer and Krusche, 2013) but no clear categorisation scheme has emerged. One additional problem typically found is a shortage of criteria explanations (e.g., Charkaoui et al., 2015; Hudli et al., 2015). Furthermore, these

[1] If multiple related criteria are used, similar subcriteria are grouped for brevity reasons, e.g., energy, CPU load, and duration measurements in Corbalan et al. (2018) are aggregated to "performance (3)".

Table 1
Literature on cross-platform app development tool evaluations.

Paper	Evaluated tools	Evaluation criteria (number of subcriteria)	Focal areas
Biørn-Hansen and Ghinea (2018)	Ionic, React Native	File system access performance	Quantitative comparison
Corbalan et al. (2018)	Cordova, Corona, Native app, NativeScript, Titanium, Xamarin	Performance (3)	Resource usage and execution time of calculations and audio/video playback
Delia et al. (2018)	Cordova, Corona, Native app, NativeScript, Titanium, Web Apps, Xamarin	Performance	Execution time of calculations
Ferreira et al. (2018)	Native app, PhoneGap, Sensa Touch, Titanium	Performance of device features (2)	App scenarios with calculations as well as camera and GPS access
Jia et al. (2018)	Cordova, Native app, Titanium, Xamarin	Performance of build, rendering, and UI response	Specific combinations of platforms and cross-platform technologies
Biørn-Hansen et al. (2017)	Ionic, Progressive Web Apps, ReactNative	Performance (3)	Quantitative study using app scenario
Ciman and Gaggi (2017)	MoSync, PhoneGap, Titanium, Web Apps	Battery usage, device sensors (7)	Evaluation of energy consumption in combination with sensor usage
Lachgar and Abdali (2017)	*none* (native vs. web vs. cross-platform in general)	14 rather simple questions to be answered before developing; six criteria for the tool selection step	Two-step process: *tools* to be selected after the main approach is chosen
El-Kassas et al. (2017)	ICPMD, J2ObjC, MD^2, MoSync, PhoneGap, Titanium, xFace, XMLVM	Tool architecture, platform support, app type, license	Variety of development approaches
Que et al. (2017)	Cordova, Native app	Development support (6), device features (5), performance (6)	Quantitative tool comparison
Vilček and Jakopec (2017)	Ionic, PhoneGap, NativeScript, Native app	Platform support (3), development support (3)	Qualitative comparison
Ahti et al. (2016)	PhoneGap	Starting duration, memory usage, app size, user experience, appearance, development support	Quantitative and qualitative evaluation criteria
Botella et al. (2016)	Ionic, Sencha	User (functionality, UI, platform support), developer (developing time, reuse, native access)	Qualitative tool comparison
Latif et al. (2016)	*none* (cross-platform approaches in general)	Scalability and maintainability, device features, resource consumption, security, IDE	Criteria definition and variety of development approaches
Rieger and Majchrzak (2016)	PhoneGap, Web Apps	Infrastructure (7), development (11), app (9), usage (4)	Criteria weighting
Umuhoza and Brambilla (2016)	13 research frameworks, 4 commercial solutions	Development process, app layer, development technique, platform support	Model-driven approaches
Charkaoui et al. (2015)	*none* (cross-platform approaches in general)	Targeted public, programming language, app type	Qualitative comparison of cross-platform approaches
Ciman and Gaggi (2015)	PhoneGap, Titanium	Battery usage, device resource usage	Evaluation of battery usage
Dhillon and Mahmoud (2015)	Adobe Air, MoSync, PhoneGap, Titanium	Platform support, license (2), development environment (8), user experience (6), functionality (29), monetization (4), security (2)	Performance benchmarks and development experience discussion
Hudli et al. (2015)	AngularJS, HTML5/JS, jQuery Mobile, PhoneGap, RhoMobile, Sencha Touch	Platform support (4), development support (7), deployment factors (6)	Criteria definition and qualitative tool comparison
Ciman et al. (2014)	jQuery Mobile, MoSync, PhoneGap, Titanium	License, community, API, tutorials, complexity, IDE, devices, GUI, knowledge	Qualitative tool comparison for apps with animations
Dalmasso et al. (2013)	jQuery Mobile, PhoneGap, Sencha Touch, Titanium	Platform support, rich user interface, backend communication, security, app extensions, energy consumption, device features, license	Performance evaluation (memory, CPU, energy consumption)
Humayoun et al. (2013)	MoSync, Native app, Titanium	Responsiveness	Qualitative user evaluation
Sommer and Krusche (2013)	PhoneGap, Rhodes, Sencha Touch, Titanium	Functionality (8), usability (6), developer support (4), reliability/performance (4), deployment (8)	Criteria definition and qualitative tool comparison
Vitols et al. (2013)	Cordova, PhoneGap, RhoMobile, Titanium	Platform support, framework development activity/maturity (3), license, device features (11)	Quantitative comparison
Xanthopoulos and Xinogalos (2013)	*none* (cross-platform approaches in general)	Distribution, programming languages, hardware & data access, user interface, perceived performance	Criteria definition
Heitkötter et al. (2012)	PhoneGap, Titanium, Web Apps	Infrastructure (7), development (6)	Foundational cross-platform criteria catalogue for this work
Ohrt and Turau (2012)	9 commercial frameworks	Developer support (8), user expectations (6)	Criteria definition and challenges of cross-platform development
Palmieri et al. (2012)	DragonRad, MoSync, RhoMobile, PhoneGap	Platform compatibility (2), development features (4), general features (4), device APIs (17)	Qualitative tool comparison
Ribeiro and da Silva (2012)	Canappi mdsl, DragonRAD, mobl, PhoneGap, Rhodes, Titanium	Technology approach, platform support, development environment, app type, device features (5)	Criteria definition

inconsistencies are also reflected in the lack of measurable metrics for the respective criteria.

It can be summed up that many authors set out to conquer the field of cross-platform app development. Without doubt, the papers shown in Table 1 provide substantial contributions. However, both the rapid evolution on the mobile device market and the proliferation of individual frameworks in the field of cross-platform development thwart the process of theory-building and mandate further work. This is also illustrated by many recently published papers that – more or less isolated – address distinct issues also discussed in this article. To conclude the study of related work, we highlight such works that address novel mobile devices.

2.2. Novel app-enabled devices

Only few years ago, mobile app development exclusively focused on frameworks and applications for smartphones and – sporadically – tablets. Nowadays, many more devices have become app-enabled and to some extent mobile, ranging from Internet-of-Things (IoT) functionalities in tiny units to self-driving vehicles.

In previous work, we have presented an initial taxonomy for the variety of consumer devices whose functionality is extensible by third-party apps already today or will be in foreseeable future (Rieger and Majchrzak, 2018). While formerly it was possible to categorise devices by operation system or hardware features, this approach is not feasible anymore: for example, the Android platform spans multiple device classes. Instead, the taxonomy distinguishes device classes according to the dimensions of *media richness of inputs*, *media richness of outputs*, and *degree of mobility* (Rieger and Majchrzak, 2018). Within each of these device classes, various devices and platforms have emerged. Whereas Android and iOS have divided most of the smartphone market amongst themselves (Forni and van der Meulen, 2017), competition among novel mobile device platforms is high and no clear winners are foreseeable. Therefore, these devices pose similar challenges for app developers compared to the beginning of smartphones several years ago (Heitkötter et al., 2013a). An overview of scientific work on apps for several novel device classes according to this classification is presented next, together with existing literature on cross-platform development approaches.

Smart TVs are on the rise worldwide, with all major manufacturers offering such devices. As a consequence, more than 90% of connected TVs sold in Germany support the HbbTV standard that has evolved from previous approaches such as CE-HTML and Open IPTV (Statista Inc., 2018; HbbTV, 2018). In the U.S., app-enabled smart TVs are already present in 35% of households (Statista Inc., 2018). Many platforms have emerged in practice, for example the open-source media centre Kodi/XBMC with various forks, Android TV, Tizen OS for TV, and webOS (XBMC Foundation, 2018; Google LLC, 2018b; Linux Foundation, 2018; LG Electronics, 2018). However, app development is often tied to a specific TV manufacturer and reflects the fragmentation in the market. Interestingly from a cross-platform perspective, many smart TV frameworks natively support app development using web technologies such as HTML5 and JavaScript, thus being well-suited for cross-platform approaches. So far, scientific research often concentrates on specific sub-topics such as interactive ads (Perakakis and Ghinea, 2015a), serious games (Ryu et al., 2014), and 3D content (Perakakis and Ghinea, 2015b) across multiple smart TV platforms.

Regarding *smartwatches*, which now have found more than just a niche in the market (Rawassizadeh et al., 2014), Google and Apple again compete for dominance with their respective Android Wear (now Wear OS) and watchOS platforms. Further players in this field are Tizen OS and webOS (Bouhnick, 2015). Some vendors have open-sourced their operating system (e.g., Android, Tizen, or webOS); however, few truly vendor-independent platforms such as AsteroidOS exist (Revest, 2018). To complicate matters, smartwatches are usually paired with a smartphone (Doud, 2015), e.g., for performance reasons (Liu and Lin, 2016) and to benefit from Internet connectivity (the latter being a specific challenge, cf. Ahola, 2015). Smartwatch apps often rely on the respective smartphone companion app; thus, cross-platform development approaches must support each combination of host and watch platform. However, some smartwatch platforms recently added stand-alone capabilities on supported devices, for instance since the launch of Android Wear 2.0 in early 2017 (Google LLC, 2018j).

In a wider sense, *wearables* such as fitness trackers are often tied to proprietary platforms, e.g., Microsoft Band (Microsoft Corp., 2018). Whereas those devices usually support pairing with different smartphone platforms, third-party app development is still limited. Vendors such as Fitbit and Garmin do not even produce devices with modifiable operating system (Bouhnick, 2015). Scientific work on wearables is therefore scarce (Kim et al., 2016). Some authors have proposed middleware approaches to span a broad range of devices (Chmielewski, 2013), in one case even on the hardware layer (Zhang et al., 2011).

Despite the vagueness of the terms, *smart home* and *IoT* devices could be a future domain for cross-platform research (Jie et al., 2015). Several open-source and closed-source platforms exist that try to attract app developers and claim to integrate a plethora of devices. Qualcomm's AllJoyn, Intel's IoTivity, Apple HomeKit, and Google Brillo are the most important players that try to establish their middleware as comprehensive solutions (Carter, 2015). For *home automation*, a host of proprietary solutions exist with a variety of application targets (Silva et al., 2012). Whether existing industry standards such as KNX can form the backbone of IoT-enabled smart homes remains to be seen. Transitions towards hybrid systems (Lilis et al., 2017) and gateway usage (Fantacci et al., 2014) will possibly solve the challenges regarding hardware but may complicate app development further.

Concerning the upcoming generation of connected cars, four main approaches for developing *in-vehicle apps* exist (Schuermans and Vakulenko, 2014). First, Android Auto, Blackberry QNX, and Windows Embedded are technologies that are rebranded by car manufacturers and run native apps on the car's head unit. Second, some cars provide a remote application programming interface (API) to allow access and control of features such as door locks. For instance, General Motors, Airbiquity, and an unofficial API for Tesla cars make use of this approach (Dorr, 2018). Third, platforms including Apple CarPlay and the MirrorLink alliance use screen mirroring, i.e., the app runs externally on the smartphone and is displayed on the car's screen (Durach et al., 2013). This approach was established due to security concerns in order to avoid executing app code on the car's main hardware. Fourth, Dash Labs, Mojio, and Automatic connect to the on-board diagnostics port to interact with the car (Dash Labs, Inc., 2018; Moj.io Inc., 2018; Automatic Labs, 2018). Although this approach requires a Bluetooth dongle as additional hardware, support is given for many cars that were not designed to be app-enabled in the first place.

Besides the underlying development approach, several papers focus on usability issues (Quaresma and Gonçalves, 2014) and "remote" human machine interfaces (HMI) (Durach et al., 2013) for the specific challenges of in-vehicle apps. For example, experimental implementations of novel concepts such as a route planning app for head-up displays (HUD) (Noreikis et al., 2014) and other potential in-vehicle apps (Wolf, 2013) are explored. To the best of our knowledge, no cross-platform framework currently exists due to the novelty of the field as well as a lack of platform accessibility from the fight for dominance between car manufacturers "owning" the platform (Schuermans and Vakulenko, 2014). Current

works deal with an *Open Service Cloud* for cars (Deindl et al., 2015) and the integration of non-automotive applications into the automotive environment (Rodriguez Garzon and Poguntke, 2012). Potentially, also a middleware approach (Deindl et al., 2015) might be an option to bridge different approaches and at the same time mitigate security risks.

This overview of novel mobile device platforms shows similar characteristics of fragmentation as the smartphone market experienced several years ago. Moreover, interactions between different device classes result in an exponentially growing amount of combinations, causing additional complexities to consider for app development. Some platforms such as Android and Tizen have branches that run on multiple devices from smart TVs to wearables, potentially simplifying the future development across device class borders. Samsung TOAST is an early initiative to simultaneously develop for Samsung Smart TV, the new Tizen platform and browsers, based on the established Apache Cordova framework (Samsung, 2018). However, barely any approach covering multiple device classes currently exists in literature or practice, with the notable exception of the gaming domain. Unity3D (Unity Technologies, 2018), one of the best-known game engines for 2D/3D games (and one even targeted by scientific research, as exemplarily illustrated by the works of Xie (2012) and Messaoudi et al. (2015)), supports 29 platforms including smartphones, smart TVs, consoles, and augmented reality devices.

3. Criteria catalogue

In the following, we describe our catalogue of criteria, which marks the foundation of our evaluation framework. We start by illustrating fundamental considerations. Then, the four perspectives of the framework – *infrastructure, development, app*, and *user* – are explained in detail.

3.1. Fundamental considerations and structure

Our aim for this paper is to propose a future-proof, long-lived, adaptive evaluation framework for cross-platform app technology. It would be a presumptuous attempt to create such a framework from scratch. Thus, the structure and the selection of criteria is based on extensive prior work, as illustrated in Section 2. Moreover, we give a rationale for criteria that we added or that we use in an extended way in comparison to existing evaluation frameworks. This follows specific literature, as far as such works are available. Alternatively, we argue for such criteria based on our experience in working on cross-platform app development frameworks. We will revisit literature gaps as part of the discussion later in this article.

Consequently, the proposed criteria are the result of a process. First, we created a synopsis of existing approaches. Then, this synopsis was extended and revised. Thereby, our criteria catalogue not only caters for the latest developments in the field but also benefits from increased flexibility and versatility. Combined with the weight profiles as explained in Section 4, we are confident in being able to set the standard for future evaluation activities.

Most notably, we categorize our criteria. Instead of presenting one large catalogue, we summarize criteria by the *perspective* on development they take. Perspectives mark a specific view on the aspects being evaluated. They thereby provide coherence: although *all* criteria are important when evaluating a framework, those that have been grouped into the same perspective are stronger related to each other than those that we put into different perspectives. Not only does this foster the comprehensibility of the criteria, but also the weighting is much easier (as will be seen in Section 4).

The consideration of different perspectives is already found in the often-cited paper by Heitkötter et al. (2013a). We have extended the original two perspectives (infrastructure and development) to four:

- *Infrastructure*: Using a cross-platform app development framework is inherently bound to preconditions. Typically, ramifications arise regarding the life cycle of developed apps. This can be summarized as the infrastructure a framework provides. Most fundamentally, this concerns the supported target platforms. Moreover, aspects of licensing, usage, and long-term prospects are considered.
- *Development*: A cross-platform framework is only as good as you can use it for developing apps. Frameworks may offer further built-in development support that can make development more rapid, support inexperienced developers, or both. Being proper for development is bound to a host of criteria that all have a technical appeal. Development criteria are those that programmers and software engineers will ask for, other aspects notwithstanding.
- *App*: Naturally, the actual app denotes whether development was successful. If an app is developed using a platform's native framework, it has access to all device features regarding sensors as well as user input and device output. A development framework should ideally provide a near-native range of support for device features such that access is versatile and easy to employ. Also, the integration of business concerns with regard to an app as a product can be subsumed by this perspective. A good example for this is security, which is considered to be very important while becoming increasingly harder to overview for developers due to its multi-faceted nature (Watanabe et al., 2017).
- *Usage*: An app is more than the sum of its functionality. Therefore, the usage perspective comprises many aspects that in systems' design would fall under the non-functional (or: quality) requirements. Besides management aspects, this perspective embodies performance characteristics and how user-friendly an app is, including considerations of aesthetics, ergonomics, and efficiency.

We deem this distinction into categories not only helpful for assessing a framework with different aspects and stakeholders (such as developers, managers, and users) in mind but also to support different devices. As already argued, the devices found in modern mobile computing are no more limited to smartphones and – technologically rather similar – tablets (Rieger and Majchrzak, 2018). The distinct app perspective (compared to Heitkötter et al., 2013a) leads to more clarity with regard to the development outcome which might differ significantly across different device classes, whereas the development itself might be similar. Thus, perspectives offer an easier way to tailor an assessment to the desired device category: In some cases, assessments might be very broad, in some very narrow. And, as we will argue later, also cases such as "good smartphone support is mandatory, but compatibility with smartwatches would be nice" can be designed. Also, the additional usage perspective focuses on cross-cutting concerns such as usability and performance which largely affect user acceptance and joy of use.

Which devices are to be targeted – or, in other words, which role multi-device support plays – is merely one aspect when thinking about case-based assessment of development frameworks. The underlying development paradigms might be to some degree tailored to more or less specific use cases. For example, cross-platform development for business apps has been discussed (Majchrzak et al., 2015b). Likewise, a focus on consumers or mobile gaming is imaginable. These cases would even combine aspects of intended usage with those deriving from multi-device support. We will further elaborate on cases when explaining the weight profiles and in the discussion.

The following four subsections explain the criteria of the perspectives in detail. Besides explaining what should actually be measured respectively expressed by a category, we also give its rationale. Whenever possible, this is done based on the literature. We refer both to evaluation papers mentioned in Section 2 and to additional work specific to the very criterion. The only work not explicitly cited is that by Heitkötter et al. (2013a). As the trailblazer for cross-platform evaluation frameworks, it contained 14 criteria of our framework already, even though in a premature form from today's perspective. Extending the already thorough work of a previous conference paper (Rieger and Majchrzak, 2016), the criteria catalogue has been refined by consulting experts in this field (cf. Section 6). Resulting from an iterative process, we have reworked the criteria descriptions to sharpen their scope (e.g., covering multiple aspects of robustness (A9) instead of a limited focus on degrading functionality), apply precise terminology (e.g., "user authentication" (U4) instead of the too general term "user management"), and eliminate potential overlaps (e.g., discerning the fields of internationalisation (I6) from the subsequent app distribution (I4)). Also, two new criteria have been added in order to better suit the needs for large-scale app development through configuration management (D8) and incorporate app development for the multitude of new mobile devices (A10). For addressability, all criteria are numbered in the form *Xy* where *X* is a character denoting the perspective (I, D, A, U) and *y* a continuous number for the specific criterion.

The last subsection is followed by Table 2 (p. 11–12 onwards). While we reference related literature – particularly the related evaluation articles compiled in Table 1 – directly for each introduced criterion, this table provides a compilation of similarities of terminology. For each criterion, we state the related work that proposed a criterion by the same term. Moreover, we name terms used with a similar meaning to our criterion. The table not only means to better relate our contribution to the existing literature but also to identify ambiguities – not all criteria must always be referred to with the same term. In addition, some authors proposed criteria that are subsumed by ours, with the term thus only appearing in the detailed description of the criterion. The table can also help to identify weakly delimited terms that are used for multiple criteria (typically *overloaded* terms such as *operating system*) as well as super terms (e.g., *features*, which can mean hardware feature, system feature, or both).

3.2. Infrastructure perspective

(I1) License: Particularly for commercial development, a framework's license is important. This question is often raised but not only relevant for open-source software (Dalmasso et al., 2013; Ciman et al., 2014; Palmieri et al., 2012). Moreover, a framework might be restrictive with regard to the usage of developed apps. While it typically is most important to consider the terms for apps developed by using the framework, license particularities regarding the framework itself can also play a role. Consider, e.g., that the long-term feasibility (I7) of a framework is limited. If the license is liberal concerning modifications of the framework and an adopting company has the resources and willingness to put effort into maintenance of it, the impact of a questionable long-term feasibility might be reduced. As part of the licensing, the pricing model needs to be considered (Hudli et al., 2015; Sommer and Krusche, 2013). Open-source frameworks are typically distributed freely under varyingly permissive regulations; a premium might be charged for maintenance and consultancy (Fitzgerald, 2006). For-payment frameworks might have a flat fee or either one-time or regular payments bound to certain conditions such as the number of developers, developed apps, and so on.

(I2) Supported target platforms: The reason for using a cross-platform framework is to provide apps for several platforms while developing only once. Consequently, the supported platforms are a major concern (Ciman et al., 2014; Palmieri et al., 2012). This remains true with Android and iOS essentially dividing the market for smartphones and tablets among themselves (Forni and van der Meulen, 2017). Widening cross-platform app development to further device categories might in fact increase the number of attractive platforms again (Rieger and Majchrzak, 2018). In addition, two versions of a platform might be different enough to consider developing for them to be similar to developing for two distinct platforms (major versions often introduce breaking changes to internal APIs as well as interface and design guidelines, e.g., Google LLC, 2018c). This, again, is particularly relevant when considering different device categories. Typically, recent versions of platforms provide novel features exploited (only) by flagship devices. These might be heavily marketed – consider, e.g., Samsung's Edge displays (Samsung, 2014) –, wherefore support is important to reach early adopters (Beal and Bohlen, 1957). At the same time, many users will not adopt new devices, thereby not frequently getting platform upgrades – or none but for a few security upgrades. This problem is worsened by the update behaviour of device vendors who, particularly for Android, maintain *forks* widely compatible to the official release but augmented with custom user interfaces and apps (Dobie, 2012). The situation is likely to become direr in the near future, with markets in developing countries being entered. Inexpensive low-end devices might quickly scale up in those markets but note that *current* devices in several markets *cannot* run the same version of a platform for reason of capabilities (see for example work by Donner (2008), Pénard et al. (2012), and Mir and Dangerfield (2013)). A final consideration are combined apps that bridge more than one device class. Such an app could, e.g., be designed for a *second screen* and support both smart TV and tablet (Neate et al., 2017), or serve as companion app such as for smartphones and smartwatches in order to offload computation, use alternative input and output capabilities, or simply cater for different user preferences.

(I3) Supported development platforms: Even though apps are not normally developed on the platform they are designed for, multiple possibilities can be encountered. Custom business logic and advanced configuration of the apps can be expressed to different extents, possibly differing from the actual app specification (e.g., domain-specific notations) using one or multiple interoperable programming languages. In addition, some degrees of flexibility play a particular role if teams are heterogeneous, i.e., developers use specific hardware and software (Palmieri et al., 2012). Software in this sense does not only comprise the operation system (with Microsoft Windows and Apple MacOS being the typical choices) but also development tools including the development environment. I3 thereby is related to D1 (development environment), although the latter concerns the integrated development environment (IDE) typically used (and often enhanced) for a framework. A good development platform support is furthermore beneficial for integration with additional app development tasks. For example, user interface (UI) and UX design might benefit from multi-platform support (Bishop, 2006).

(I4) Distribution channels: For the majority of users, there are only few ways to acquire news apps for their devices. Typically, platform- or vendor-specific app stores provide large repositories of apps, such as the Apple App Store and Google Play (Jansen and Bloemendal, 2013). As users are accustomed to searching for apps on these platforms, it is essential to support the proprietary stores to reach a high number of users. While cross-loading of apps technically is easy, vendors might hide this functionality for strategical reasons, and to make sure users do not compromise

their own safety. Therefore, broad support for the relevant stores is desirable. While this might seem as naturally given, not all kinds of apps can necessarily be uploaded to all stores. One example are PWAs, progressively-enhanced responsive web sites that are discoverable via search engines and provide app-like interactions using modern web technologies for offline capabilities, content updates, and notifications (Russell, 2015; Majchrzak et al., 2018; Biørn-Hansen et al., 2018)). While explicitly designed for mobile devices, PWAs cannot be installed via traditional app stores. Also, cross-platform frameworks differ in the degree of compatibility with app store restrictions and submission regulations (Sommer and Krusche, 2013; Dhillon and Mahmoud, 2015). Particularly well integrated frameworks may support features such as the rating of apps to improve app store ranking as well as roll-out support for updates (Hudli et al., 2015).

(I5) Monetisation: Most apps have neither been created with purely philanthropic purposes nor merely for the joy of programming – although such apps surely exist (Jakuben, 2013). Therefore, the monetisation possibilities of apps created with a certain framework need to be assessed. There are several possibilities, which might also be used in combination (Dhillon and Mahmoud, 2015). Tang (2016) distinguishes four major monetization models, and we add free apps as a fifth category of apps with specific business purpose:

- *Paid*: Apps can be sold for a one-time fee before downloading or after a limited test period. This is typically done using the proprietary stores (see I4) by using their integrated payment options.
- *Freemium*: If apps follow a freemium model, they can be downloaded and used for free, but users need to pay if they want to have access to advanced features or full content and services after reaching a predefined usage threshold. This model is often employed in games (deprecatorily coined *pay-to-win* (Alha et al., 2014) if used excessively), where players for example will progress quicker when buying items for actual money (Hsiao and Chen, 2016). The payment is usually performed via *in-app-purchases*, in case of games often consisting of very small payments (*micropayment*) per feature or upgrade.
- *Paidmium*: This model combines paid downloads and in-app purchases, usually found in complex apps such as navigation. Although not always paid for the initial download, subscription-based models are a form of paidmium as they are not usable without login to a paid account and necessitate a regular payment in order to retain access to the app's full functionality.
- *In-app advertising*: Advertisements can be shown as part of the usage of the app. There are ample possibilities how this can be done (including banner ads, sponsored content, and white-labelling of the app itself) and to which degree the advertisements interfere with the usage of the app (Li et al., 2018).
- *Free*: Especially business apps (Heitkötter et al., 2015) may be offered for free to potential users while simultaneously serving a specific business purpose. This includes information portals to increase customer satisfaction as well as additional services (e.g., apps for mobile banking or service booking). They provide value to mobile users and improve customer loyalty, besides fostering process automation (as even studied before the emergence of the widened possibilities through mobile computing (Meuter et al., 2000)).

Strictly speaking, apps might also provide features that are undesirable for users. For example, recent studies have revealed that *some* apps contain software components that are able to track ultrasonic sounds used for perfidiously tracking users (Arp et al., 2016). While such means might offer a source of data monetisation, we exclude it from further considerations since we deem it ethically indefensible.

Development frameworks may or may not support means of monetisation and they might offer particular good support for some of them. Such features need to be judged in the light of direct costs and omissions to the app store operator (see I4 (Distribution Channels)). Good support includes interfaces to payment providers, pre-designed functionality for in-app payments, support for various types of advertisements, and access to advertising networks (Dewan and Chen, 2014; Google LLC, 2018e).

(I6) Internationalisation: Apps are typically distributed globally. Even if only one language version is available, there are normally no restrictions regarding who can install an app. There might be specific reasons to restrict users to local versions or to even prevent the distribution in certain geographic regions. For example, legal conflicts and national legislation may prohibit the distribution in parts of the world (as reported by Ng et al. (2014) for China). From a positive point of view, internationalisation and localisation can offer added value by broadening the base of potential users and by providing better targeted functionality. Localisation can be supported by the development framework. It can even go as far as built-in translation capabilities as well as an easy support of a multi-language operation mode. This is further aided if features such as conversion tools (e.g., for dates, currencies, and units) are provided (Sommer and Krusche, 2013). Additionally, frameworks might bring in support for national idiosyncrasies, e.g., API support for state-specific services, for example regarding authentication or personal data records.

(I7) Long-term feasibility: The choice of an app development approach can be a strategic decision for a commitment over multiple years. Depending on the framework, the kind of apps developed as well as their intended lifetime, and the situation in the developing company, significant initial investment might be necessary. Moreover, there can be the risk of a technological lock-in (as particularly discussed in the context of proprietary software by Zhu and Zhou (2012)). Initial investment includes market studies, assessments (which, as we hope, are much easier using our framework), fees, training materials and training courses, and recruiting. The risk of lock-in can only be partly mitigated by looking for good compatibility, adherence to standards, and the usage of well-known technologies. It is particularly high for small companies, which might lack the resources to quickly correct an ill choice and which typically will invest in just one cross-platform development framework at a time. Whether a framework is suitable for an extended period cannot be assessed in a completely objective way (you may forecast but you cannot prophesy), but maturity, stability, and activity are indicators that help with an educated judgement:

- The *maturity* of a framework can be judged according to a long-lasting existence, a large community of developers and resulting apps, as well as historic events. The latter may for example mean that it can be analysed how emergent security flaws have been handled.
- The *stability* can be particularly seen when looking at the history and future schedule of releases. At which rate have new features been introduced? Have major releases been backward compatible, and if so, how far? Are update cycles (for minor releases) sufficiently short? Are bug-fixes and security updates provided regularly and timely in case of major flaws?
- Besides release cycles, the *activity* of a framework relates to the general contributions of developers and users: Does an active community exist that reports bugs and discusses solutions to these issues? Is this community likely to provide support where official documentation falls short? Does this

community probably even support the future development of the framework when a major backer withdraws? Particularly in case of open-source products not backed by a large corporation, a healthy community might even blend with the development team.

Moreover, if a framework is supported, led, or even owned by a company or a consortium, the reputation of the key stakeholders should be scrutinized. Typically, financial or even technological support (such as code contributions) by commercial entities is particularly valuable for open-source frameworks (cf. the work of Andersen-Gott et al., 2012). Additionally, news, plans, and rumours can be checked. For example, the announcement of a new framework by a company might eventually mean the demise of its predecessor. Likewise, changes to fundamental technology (e.g., a JavaScript engine) could mean that a framework is strengthened or becomes obsolete. Technology breakthroughs may have the same effect - as could happen with WebAssembly (Wagner, 2017). Finally, it should be considered whether support inquiries require a premium. Such costs might not necessarily be considered negative; in fact, they may hint to a good outlook particularly in the case of open-source software for which commercial "premium support" exists to help with development issues (Hudli et al., 2015; Sommer and Krusche, 2013).

3.3. Development perspective

(D1) Development environment: Rapid development is typically supported by the use of an IDE. The maturity and feature-richness of IDEs can greatly influence development productivity – sometimes also negatively when usability challenges of managing too much functionality overburdens users (Kline and Seffah, 2005). Features such as auto-completion and the integration of library documentation help with the actual coding. Built-in debuggers and emulators support a rapid app development cycle (Hudli et al., 2015; Sommer and Krusche, 2013; Ciman et al., 2014; Palmieri et al., 2012; Dhillon and Mahmoud, 2015). If a certain IDE is not enforced by the cross-platform framework, and in particular if there is freedom with regard to accustomed workflows, the initial effort of starting to work with a framework can be significantly lowered. This can lower the set-up effort of dependencies such as runtime environments or SDK (Sommer and Krusche, 2013).

(D2) Preparation time: Apps are typically developed rapidly. Thus, the realized learning curve should be favourable, reflecting rapid subjective progress of a developer in getting acquainted with the capabilities of a framework. The entry barrier is also influenced by the required technology stack and the number and kind of supported programming languages (Xanthopoulos and Xinogalos, 2013; Ciman et al., 2014; Sommer and Krusche, 2013; Palmieri et al., 2012). Being able to rely on well-known programming paradigms can further reduce the learning efforts needed before being able to work productively (Ciman et al., 2014). Moreover, the documentation of the API is important – particularly, if a framework poses unique characteristics or novel ways of providing common functionality. Additionally, "Getting started" guides, tutorials, screencasts, and code examples make a framework more accessible and help to clarify features and idiosyncrasies; a corpus of best practices, user-comments, and technical specifications helps with staying productive once an approach is initially conquered (Sommer and Krusche, 2013; Dhillon and Mahmoud, 2015).

(D3) Scalability: Particularly in large-scale or distributed development projects, apps need to scale. For this purpose, proper modularisation is needed. The app structure is heavily influenced by the general possibilities for partitioning into subcomponents and by architectural conditions. For example, using the widely applied Model-View-Controller pattern has profound ramifications for other design decisions. Ideally, more developers can be added to a project while the app's functionality grows (Hudli et al., 2015; Palmieri et al., 2012). A framework that supports modular or even component-based development can support this division of labour – or even guide it. Moreover, when layering is supported and components can be given specified interfaces and interaction, a higher level of specialisation is possible for developers. Besides adding to the scalability, this might have a positive impact on software quality.

(D4) Development process fit: From the traditional waterfall approach (Royce, 1970) over integrated methodologies such as the Rational Unified Process (Jacobson, 1999) till the variety of agile methods, many ways of developing software are employed. Although all methodologies have common characteristics (Dyck and Majchrzak, 2012), actual development differs widely. Compare, for example, the design-heavy waterfall approach to Extreme Programming (Beck, 1999). Consequently, a framework should be compatible with custom ways of developing software. As the first step, it can be scrutinized how much effort is required to create the *minimum viable product*. Frameworks differ with regard to the initial configuration that has to be made, so-called boilerplate code, and the following effort for incrementing the scope. Thereby, D4 is also related to D3 (Scalability), as the organisational aspect of specialisation, which might be fostered by methodology-fit, influences the scalability in terms of functionality. Tailored views and specialised tool can support modularising development with a profound role concept, contrasting the work of full-stack developers typically encountered in small projects (Wasserman, 2010).

(D5) User interface design: The UI design is essential when developing user-centred application, which most apps are. At the same time, the input and output heterogeneity of mobile device hardware (A4, A5) poses challenges to the development approach of mobile UIs using either flexible descriptions such as responsive designs or multiple layouts for specific ranges of screen sizes (Eisenstein et al., 2001; Rieger and Kuchen, 2018b). Commonly, not all cross-platform frameworks put weight on platform-agnostic UI aspects, partially leaving it at individual implementations per target platform (Google LLC, 2018f). Graphical user interfaces are usually specific to a platform and in many cases only covered by a default appearance defined by the framework (Heitkötter et al., 2013b). Depending on development requirements, a separate *What You See Is What You Get* (WYSIWYG) editor can be very helpful. Such editors can be used to design appealing, ergonomic interfaces for multiple devices. They can also increase the pace of development compared to repeatedly deploying the full app to a device or an emulator. At the same time, reasonable support for platform-adequate designs without too much effort from developers is preferable, for instance considering round and rectangular layout types for smartwatches.

(D6) Testing: User interface, business logic, and possible additional components of apps need to be thoroughly tested (Hudli et al., 2015; Sommer and Krusche, 2013). In addition to the well-known techniques and approved strategies of testing desktop and server applications such as unit tests, the context-sensitivity of mobile devices should be honoured. For this purpose, mobile scenarios (such as moving around, or getting a phone call while using an app) need to be considered and external influences (such as varying connectivity) could be simulated (Majchrzak and Schulte, 2015). In addition to this, monitoring the app at runtime can further improve its testability. This includes, e.g., providing a developer console, meaningful error reporting, and logging functionalities for app-specific and system events. Also, remote debugging on a connected device rather than using emulator environments allows for more realistic test results. Additional tool support can aid testing further and provide test coverage visualisation and metrics to support test controlling (Hudli et al., 2015).

(D7) Continuous delivery: Life cycle support does not end with testing but should also include deployment. Being able to rely on a solid toolchain greatly simplifies the deployment. For example, a framework may leave you with source code generated for the target platforms, may support the generation of native apps, or may go all the way by providing signed packages, possibly even supporting developers in deploying these to devices or app stores. Frameworks vary greatly, from requiring each target platform's native SDKs to using external build services and cloud-based approaches (Hudli et al., 2015; Sommer and Krusche, 2013). Particularly if following an agile method, *continuous delivery* platforms to automate building, testing, and deploying an app may be used. Frameworks can be explicitly designed to integrate with such toolchains, for example by generating project-specific scripts. Additionally, a framework might offer advanced build options (such as code *minification*) and continuous app store integration (e.g., for automatically publishing updated versions) (Hudli et al., 2015).

(D8) Configuration management: Often, apps do not exist in isolation but have multiple versions when considering multiple roles (such as user and administrator with different capabilities), theming (or branding for white-label apps), and significant regional peculiarities (e.g., right-to-left script). Depending on available app store features, developers in addition might need to supply different app packages for free and paid versions with varying functionality. Cross-platform frameworks can also support the development of such feature variations similar to product lines, either by providing different app packages or by allowing a dynamic transition between versions without re-installing the app.

(D9) Maintainability: The application life cycle does not typically end with one-time deployment (D7). Rather, software is maintained for a shorter or longer period, over which the code base evolves (Sommer and Krusche, 2013). Maintainability generally is hard to quantify. Although simple metrics such as lines of code (LOC) can give a basic idea (the more source code the harder to maintain), more complex metrics might reveal a different picture, for instance when considering code complexity (cf. with the work of Gill and Kemerer, 1991, but also with the critical assessment of Shepperd (1988)). While such metrics can be used to compare an app against reference apps, qualitative aspects need to be taken into account. This concerns readability of code, use of design patterns, the kind of in-code documentation and similar aspects; possibly in conjunction with the amount of training, familiarisation, and other preparatory efforts (see also D2). These considerations are similar to the discussion about programming languages, where so-called *gearing factors* are used to compare the amount of code per unit of functionality (QSM, 2009). It is problematic to apply advanced maintainability metrics due to the heterogeneity and varying complexity of frameworks. This counts even stronger for generative approaches and the resulting diversity of platform-specific programming languages. As an additional aspect, the reusability of source code across development projects can be evaluated as well as the portability to other software projects (Sommer and Krusche, 2013).

(D10) Extensibility: Although a framework should cover a certain scope and enable the development of *typical* apps within that scope, project-specific requirements may go beyond the provided functionality. However, if they are unlikely to be added as features due to a low general priority, a framework's extensibility becomes important. It can be more flexibly used if custom components can be added and third-party libraries can be included. Typically covered areas include extensions for the UI (such as alternative or additional widgets), access to device features, and libraries for common tasks such as networking and data transfer (Hudli et al., 2015; Palmieri et al., 2012).

(D11) Integrating custom code: Some applications require native code or third-party libraries to be run. While this seemingly contradicts the principle of cross-platform development, it can be necessary in some cases. This applies in particular when the desired functionality cannot be realized using extensions (D10). Using native platform APIs might enable access to platform functionalities and device features that are not currently supported by a framework or unique to a platform (Sommer and Krusche, 2013; Palmieri et al., 2012). Moreover, companies might want to integrate native code to reuse functionalities, for example when successively migrating apps that were developed natively to a cross-platform approach.

(D12) Pace of development: While many of the above criteria have an influence on how rapid development will be, there are some particularly influencing factors. Especially the amount of boilerplate code necessary for functional app skeletons (cf. Heitkötter et al., 2014) and the availability of pre-defined functionality for typical requirements (such as user authentication) facilitate swiftness. Ignoring possible salary differences based on programming language proficiency, the overall development speed has direct influence on the variable costs and, ultimately, the return-on-investment.

3.4. App perspective

(A1) Access to device-specific hardware: While today's devices possess high processing power, their hardware features – especially sensors – account for the versatility and ubiquitous use. In consequence, access to platform- and device-specific hardware is vital for cross-platform frameworks (Hudli et al., 2015; Dalmasso et al., 2013; Ciman et al., 2014; Sommer and Krusche, 2013; Dhillon and Mahmoud, 2015). Frameworks with poor coverage incur the risk of feature-poor apps to be developed. A plethora of device hardware is present today, including sensors such as camera, microphone, GPS, accelerometer, gyroscope, magnetometer, and temperature scale as well as novel additions such as a heart rate monitor. Moreover, devices (more precisely: cyberphysical systems) may also offer bidirectional interaction through actuators, enabling them to modify their environment (e.g., in smart home apps).

(A2) Access to platform-specific functionality: Similar to A1, apps can only make use of the versatility of modern mobile devices if frameworks provide access to the possibilities they offer. Such platform-specific functionalities include a persistence layer, providing file system access and storage to a database, contact lists, information on the network connection, and battery status (Hudli et al., 2015; Dhillon and Mahmoud, 2015). Furthermore, support for extending the app with complex business logic using general-purpose programming languages might be required in specific app projects but may be inherently restricted by the framework's paradigm of development. In-app browser support can make development much easier when web-based content is accessed (Hudli et al., 2015). Background services can serve for the realisation of continuous feature execution such as push notifications and monitoring (Sommer and Krusche, 2013).

(A3) Support for connected devices: In addition to accessing hardware and platform functionality (A1, A2), the support for connected devices can be scrutinized. Wearables and other small mobile devices as well as sensor/actuator networks of cyberphysical systems often rely on coupling with a master device, typically a smartphone. This enhances their capabilities or might be used for occasional synchronisation. The interaction of devices respectively the extension of capabilities through other gadgets has become more important; thus, the support of viable device combinations by frameworks should be assessed (Seyed et al., 2015). Specifically, this concerns the kind and richness of access to coupled devices, their data, and their sensors. Moreover, the provision of additional UI components should be evaluated, if applicable. The latter for example applies to smartwatches, which on a smaller screen provide

a selection of a host device's functionality. Realising the support can be trivial if a platform provides a layer of abstraction exposing the coupled device as if it was a regular device component. However, particularly due to the multitude of possible combinations and the specificity of these, cross-platform frameworks will need to provide explicit support in many cases, which brings additional complexity.

(A4) Input device heterogeneity: Mobile devices allow for a multitude of inputs. This includes traditional means such as keyboard and mouse, (multi-) touch screens, remote controls, and hardware buttons, as well as modern means such as voice recognition and futuristic input technologies using gestures or neural interfaces (Rieger and Majchrzak, 2018). However, not all devices allow all possible means of input; this is even true within one device class (consider, e.g., smartphones that understand ultrasound gestures (Horsley, 2016)). In addition, devices typically support complex inputs via several alternatives. Think, e.g., of a smartphone screen, which can be manipulated via multi-touch gestures such as taps, swipes, pinches, and pressure, but also reacts to orientation changes and hardware buttons. Cross-platform frameworks need to make these possibilities available to developers, consider the lack of input actions on individual devices, and respect platform- or device-specific patterns (e.g., scrolling on smartwatches may be achieved by rotating bezels, digital crowns, or screen swiping). Ideally, they should also provide support for simple usage – for example by providing means to register multi-touch events or more abstract user actions instead of the need to observe single touches and make sense of their combination.

(A5) Output device heterogeneity: Heterogeneity is also given for the output possibilities a device offers. Most have screens for visual output, which differ in size, resolution, format (quadratic vs. rectangular vs. round), colour palette, frame rate (e.g., very slowly updated E-ink screens), and opacity (e.g., augmented reality projections). Moreover, many other possibilities for output exist, such as projection and sounds (Rieger and Majchrzak, 2018). Adaptability is challenging for traditional devices already (Amatya and Kurti, 2014) and becomes very complicated with novel gadgets. Moreover, apps need to cope with device class specific context changes to realize well-understood design ideals (Schilit et al., 1994) such as a day/night screen mode for in-vehicle apps.

(A6) Application life cycle: The life cycle inherent to an app should be supported by a framework. This must not be confused with the development life cycle addressed in D4 (Development process fit), D7 (Continuous delivery), and D9 (Maintainability). The app life cycle comprises of starting, pausing, continuing, and exiting an app (Sommer and Krusche, 2013), as well as possible others states in dependence on the platform. Multithreading, continuously running background services, and notifications further extend the states in which an app is executed without necessarily providing a graphical UI. In addition, individual views and view elements might have divergent states or even life cycles, e.g., enforced teardown of inactive widgets on Android to reduce memory usage (Google LLC, 2018i).

(A7) System integration: Many apps rely on (business) backend systems, which is typically in the interest of the app vendors (Majchrzak and Heitkötter, 2013). Frameworks preferably offer several options for integration in existing ecosystems and workflows, including support for data exchange protocols and serialisation as well as multiple data formats (Dalmasso et al., 2013). Apps need to be able to consume web services for data storage and processing. Ideally, inter-app communication should be possible (consider a banking app requesting transaction authorisation from an identity verification app). Additionally, workflow-oriented use cases typically rely on collaboration from several user roles, which may be supported by an app framework. Finally, system integration also means that apps need to be customisable, e.g., to follow the overall design endorsed by a *corporate identity* (Sommer and Krusche, 2013).

(A8) Security: App security is an increasingly discussed topic with many facets (Watanabe et al., 2017). Frameworks can support the development of secure apps with regard to several security attributes (Parker, 1998):

- As regards *confidentiality*, mobile platforms provide means for managing access permissions regarding platform and device features. In general, such permissions should be handled restrictively. Apps should only request permissions on demand (e.g., access to contacts only if contacts are to be imported) (Google LLC, 2018h). Concerning the generally low understanding of app permissions (Kelley et al., 2012), this might raise the user awareness and acceptance of apps respecting security best practices.
- From an *integrity* point of view, sensitive data should be secured using encryption on the device file system or database. Moreover, secure data transfer protocols impede eavesdropping when communicating with backend systems and web services (Dalmasso et al., 2013; Hudli et al., 2015).
- Regarding *control*, support for user-input validation and prevention of code injections, cross-site request forgery, and similar attack patterns are preferable (Hudli et al., 2015).

A framework might provide basic or advanced support for security, ideally freeing inexperienced developers from explicitly implementing security-relevant functionality.

(A9) Robustness: Criteria A1–A5 leave much freedom to the app developer. Apps should include intelligent fallback mechanisms in case specific features are unsupported or restricted. The naïve option is to redirect a user to a web page. More sophisticated handling includes *graceful degradation* techniques such as simpler representations (Ernsting et al., 2016) and the employment of alternative functions that make up for the unavailable ones. In addition, robustness also refers to fault-tolerant and resilient mechanisms, for example by acting gracefully if permissions are denied by the user or sensors are deactivated. Fault-tolerance particularly applies to common situations with poor or unavailable Internet access. A framework should enable offline capabilities to keep apps operational in low connectivity situations, for example by storing assets and content locally on the device and caching data that needs to be sent to the backend server as soon as connectivity is re-established (e.g., Chun et al. (2012)).

(A10) Degree of mobility: Mobility considerations for the desired apps also influence the framework choice. In contrast to the related criteria on available target platforms (I2) and hardware access (A1), different degrees of mobility strongly affect app mechanics and emphasize features beyond infrastructure considerations. On a high level, four categories can be distinguished (Rieger and Majchrzak, 2018):

- *Stationary*: App-enabled devices do not need to be mobile (e.g., smart home devices). They differ from traditional desktop applications regarding input/output characteristics but barely need to consider contextual information.
- *Mobile*: Typical mobile applications must process various types of context information such as location, time, and further sensor values (e.g., ambient light for in-vehicle UIs).
- *Wearable*: In addition to the usage context, wearables need to adapt to personal preferences and unobtrusively blend with the user's life (e.g., when to propose recommendations or avoid distraction through notifications). Also, other types of sensors might require continuous event processing for applications such as health monitoring.
- *Autonomous*: The highest degree of mobility requires advanced self-adaptation capabilities for the cyber-physical

system to react to expected and unexpected situations. The app might therefore exhibit agent characteristics or apply business rules for automated decision making.

3.5. Usage perspective

(U1) Look and feel: The UI elements provided by a framework should have a native look and feel rather than resembling a web site (Sommer and Krusche, 2013). If generated apps use a truly native interface, it should be created in the way typical for a platform. Elements, views, and interaction possibilities can be evaluated according to the respective human interface guidelines provided by platform vendors (Google LLC, 2018a; Apple Inc., 2018a). Apps should also support the platform-specific usage philosophy, e.g., regarding the position of navigation bars, scrolling, and gestures (Sommer and Krusche, 2013). While form-based interfaces suffice in many cases (Heitkötter et al., 2013b) and are relatively simple to realize, richer user interfaces with 2D anima-

Table 2
Literature references to criteria and related terms.

Criterion		Literature referencing the criterion or related/subordinate terms
I1	License	(Palmieri et al., 2012; Heitkötter et al., 2013a; Dhillon and Mahmoud, 2015), (direct) costs (Dhillon and Mahmoud, 2015; Hudli et al., 2015; Heitkötter et al., 2013a; Sommer and Krusche, 2013), open-source (Hudli et al., 2015; Vitols et al., 2013; Palmieri et al., 2012), availability (El-Kassas et al., 2017)
I2	Target platforms	Supported platforms (El-Kassas et al., 2017; Dhillon and Mahmoud, 2015; Dalmasso et al., 2013; Sommer and Krusche, 2013; Vitols et al., 2013; Heitkötter et al., 2013a), mobile platforms (Vilček and Jakopec, 2017), versions (Botella et al., 2016), portability (Sommer and Krusche, 2013), mobile operating systems (Palmieri et al., 2012)
I3	Development platforms	(Programming/development) languages (Que et al., 2017; Botella et al., 2016; Charkaoui et al., 2015; Ohrt and Turau, 2012; Palmieri et al., 2012; Vilček and Jakopec, 2017; Ribeiro and da Silva, 2012; Dhillon and Mahmoud, 2015), computer operating systems (Dhillon and Mahmoud, 2015), technologies (Xanthopoulos and Xinogalos, 2013), OS support (Palmieri et al., 2012)
I4	Distribution channels	App store (Charkaoui et al., 2015; Dhillon and Mahmoud, 2015; Sommer and Krusche, 2013) publishing (Lachgar and Abdali, 2017), distribution (Que et al., 2017; Xanthopoulos and Xinogalos, 2013; Heitkötter et al., 2013a), market (Charkaoui et al., 2015) market place deployment (Xanthopoulos and Xinogalos, 2013), analytics platform (Dhillon and Mahmoud, 2015)
I5	Monetisation	Sales (Lachgar and Abdali, 2017), in-app purchases (Dhillon and Mahmoud, 2015), mobile ad platform support (Dhillon and Mahmoud, 2015)
I6	Internationalisation	(Sommer and Krusche, 2013)
I7	Long-term feasibility	(Heitkötter et al., 2013a), popularity (Lachgar and Abdali, 2017), count of updates (Vitols et al., 2013), community (Vitols et al., 2013)
D1	Development environment	(Latif et al., 2016; Heitkötter et al., 2013a; Ribeiro and da Silva, 2012), IDE (Que et al., 2017; Dhillon and Mahmoud, 2015; Hudli et al., 2015; Palmieri et al., 2012), tool restrictions (Sommer and Krusche, 2013), dependencies (Sommer and Krusche, 2013)
D2	Preparation time	Documentation/documents (Lachgar and Abdali, 2017; Que et al., 2017; Vilček and Jakopec, 2017; Botella et al., 2016), documentation completeness and quality (Sommer and Krusche, 2013), learning curve (Lachgar and Abdali, 2017), learning effort (Sommer and Krusche, 2013), community (Vilček and Jakopec, 2017), speed and complexity of installation (Vilček and Jakopec, 2017), developer and user support groups (Hudli et al., 2015), ease of development (Heitkötter et al., 2013a)
D3	Scalability	(Latif et al., 2016; Heitkötter et al., 2013a), complexity (Lachgar and Abdali, 2017), architecture (El-Kassas et al., 2017), architectural implication (Hudli et al., 2015), MVC support (Dhillon and Mahmoud, 2015; Palmieri et al., 2012)
D4	Development process fit	Development process (Umuhoza and Brambilla, 2016), architecture (Palmieri et al., 2012)
D5	UI design	GUI design(er) (Heitkötter et al., 2013a; Ohrt and Turau, 2012), graphical tool for GUI (Lachgar and Abdali, 2017), UI design assistant (Botella et al., 2016), no-code/low-code support (Hudli et al., 2015), customizability (Sommer and Krusche, 2013)
D6	Testing	(Umuhoza and Brambilla, 2016; Sommer and Krusche, 2013), debugging (Que et al., 2017; Botella et al., 2016; Dhillon and Mahmoud, 2015; Sommer and Krusche, 2013; Ohrt and Turau, 2012), simulator (Latif et al., 2016), emulator (Hudli et al., 2015; Ohrt and Turau, 2012), test framework (Hudli et al., 2015)
D7	Continuous delivery	Building time (Jia et al., 2018), build service availability (Dhillon and Mahmoud, 2015), build support (Hudli et al., 2015), simplified/automatic builds (Sommer and Krusche, 2013), compile without SDK (Ohrt and Turau, 2012), instant update (Charkaoui et al., 2015), upgrade (Que et al., 2017), updates (Hudli et al., 2015)
D8	Configuration management	Production support (Hudli et al., 2015)
D9	Maintainability	(Latif et al., 2016; Heitkötter et al., 2013a), supportability (Sommer and Krusche, 2013)
D10	Extensibility	(Sommer and Krusche, 2013), libraries (Hudli et al., 2015), app extensions (Dalmasso et al., 2013), plug-in repository (Vitols et al., 2013), plug-in extendibility (Palmieri et al., 2012)
D11	Custom code integration	Native access (Botella et al., 2016), extensibility with native code (Ohrt and Turau, 2012), native APIs (Palmieri et al., 2012), reuse (Botella et al., 2016), app layer (Umuhoza and Brambilla, 2016), access to native UI (Dhillon and Mahmoud, 2015)
D12	Pace of development	Development rate (Lachgar and Abdali, 2017), speed of development (Heitkötter et al., 2013a), developing time (Botella et al., 2016), time to market (Lachgar and Abdali, 2017), budget (Lachgar and Abdali, 2017), complexity of development (Vilček and Jakopec, 2017), easiness of development (Ahti et al., 2016)
A1	Hardware access	(Xanthopoulos and Xinogalos, 2013), device features (Latif et al., 2016), device API (Charkaoui et al., 2015), device resource support (Hudli et al., 2015), sensor data capture (Dhillon and Mahmoud, 2015), built-in features (Dalmasso et al., 2013), hardware sensors (Sommer and Krusche, 2013), mobile device functions (Vitols et al., 2013), platform-specific features (Heitkötter et al., 2013a), APIs (Palmieri et al., 2012), accelerometer (Ciman and Gaggi, 2017; 2015; Ciman et al., 2014; Dhillon and Mahmoud, 2015; Vitols et al., 2013; Palmieri et al., 2012; Ribeiro and da Silva, 2012), compass (Ciman and Gaggi, 2017; 2015; Ciman et al., 2014; Dhillon and Mahmoud, 2015; Palmieri et al., 2012), proximity (Ciman and Gaggi, 2017; Dhillon and Mahmoud, 2015), GPS (Ciman and Gaggi, 2017; Que et al., 2017; Ciman and Gaggi, 2015; Ciman et al., 2014; Dhillon and Mahmoud, 2015; Ribeiro and da Silva, 2012), geolocation (Sommer and Krusche, 2013; Vitols et al., 2013; Palmieri et al., 2012) camera (Ciman and Gaggi, 2017; Que et al., 2017; Ciman and Gaggi, 2015; Ciman et al., 2014; Dhillon and Mahmoud, 2015; Vitols et al., 2013; Palmieri et al., 2012; Ribeiro and da Silva, 2012), audio record (Ciman and Gaggi, 2017), microphone (Ciman and Gaggi, 2015; Ciman et al., 2014; Dhillon and Mahmoud, 2015), Bluetooth (Dhillon and Mahmoud, 2015; Ohrt and Turau, 2012; Palmieri et al., 2012), accelerator (Que et al., 2017), GPU acceleration (Dhillon and Mahmoud, 2015), light (Ciman and Gaggi, 2017), notification light activation (Dhillon and Mahmoud, 2015), noise cancelation microphone (Dhillon and Mahmoud, 2015), NFC (Dhillon and Mahmoud, 2015; Palmieri et al., 2012), gyroscope (Dhillon and Mahmoud, 2015), barometer (Dhillon and Mahmoud, 2015), Wi-Fi positioning (Dhillon and Mahmoud, 2015), cellular positioning (Dhillon and Mahmoud, 2015), network (Sommer and Krusche, 2013; Vitols et al., 2013), low-level networking (Dhillon and Mahmoud, 2015), connection (Palmieri et al., 2012), (hardware) buttons (Sommer and Krusche, 2013), device (information) (Palmieri et al., 2012)

(continued on next page)

Table 2 (*continued*)

Criterion		Literature referencing the criterion or related/subordinate terms
A2	Platform Functionality	Native features (Lachgar and Abdali, 2017), device resource support (Hudli et al., 2015), functionality (Botella et al., 2016), platform-specific features (Heitkötter et al., 2013a), contacts (Que et al., 2017; Dhillon and Mahmoud, 2015; Vitols et al., 2013; Palmieri et al., 2012; Ribeiro and da Silva, 2012), media (Que et al., 2017; Vitols et al., 2013; Ribeiro and da Silva, 2012), files/file system access (Dhillon and Mahmoud, 2015; Vitols et al., 2013; Palmieri et al., 2012), (user/system/push/alert/sound) notifications (Dhillon and Mahmoud, 2015; Vitols et al., 2013; Palmieri et al., 2012), calendar (Dhillon and Mahmoud, 2015; Palmieri et al., 2012), SMS (Dhillon and Mahmoud, 2015), call log (Dhillon and Mahmoud, 2015), voice activation (Dhillon and Mahmoud, 2015), native map support (Dhillon and Mahmoud, 2015), background processes (Dhillon and Mahmoud, 2015), in-app browser (Hudli et al., 2015), storage (Vitols et al., 2013; Palmieri et al., 2012), data access (Xanthopoulos and Xinogalos, 2013), local database (Hudli et al., 2015), database access (Sommer and Krusche, 2013), barcode (scanner) (Vitols et al., 2013; Palmieri et al., 2012), menu (Palmieri et al., 2012)
A4	Input Heterogeneity	touch support (Hudli et al., 2015), gestures (Sommer and Krusche, 2013), swipe, pinch (Dhillon and Mahmoud, 2015)
A6	App Life Cycle	(Sommer and Krusche, 2013)
A7	System Integration	Social APIs, Cloud APIs (Dhillon and Mahmoud, 2015), backend communication (Dalmasso et al., 2013), corporate identity (Sommer and Krusche, 2013)
A8	Security	(Lachgar and Abdali, 2017; Latif et al., 2016; Dalmasso et al., 2013), secure storage access, code obfuscation (Dhillon and Mahmoud, 2015), security vulnerabilities, encrypted local storage (Hudli et al., 2015)
A9	Robustness	stability, reliability (Sommer and Krusche, 2013)
U1	Look and Feel	(Xanthopoulos and Xinogalos, 2013; Heitkötter et al., 2013a; Lachgar and Abdali, 2017), user experience (Lachgar and Abdali, 2017; Ahti et al., 2016; Heitkötter et al., 2013a), appearance (Ahti et al., 2016), (rich) UI (Botella et al., 2016; Dalmasso et al., 2013), UI response time (Jia et al., 2018), interaction-response (Humayoun et al., 2013), user-perceived performance (Xanthopoulos and Xinogalos, 2013), intuitiveness (Ohrt and Turau, 2012), UI functionality, native UI components (Sommer and Krusche, 2013), fluidity, animations (Lachgar and Abdali, 2017)
U2	Performance	(Sommer and Krusche, 2013), execution time (Biørn-Hansen and Ghinea, 2018), duration (Corbalan et al., 2018; Delia et al., 2018), energy/power consumption (Corbalan et al., 2018; Ciman and Gaggi, 2017; 2015; Ciman et al., 2014; Latif et al., 2016; Dalmasso et al., 2013), app size (Jia et al., 2018; Ahti et al., 2016; Ohrt and Turau, 2012), size of installation (Biørn-Hansen et al., 2017; Sommer and Krusche, 2013), CPU (load) (Corbalan et al., 2018; Latif et al., 2016), CPU occupancy ratio (Que et al., 2017), RAM/memory usage (Jia et al., 2018; Ohrt and Turau, 2012; Ahti et al., 2016), memory occupancy (Que et al., 2017), application/activity launch time (Biørn-Hansen et al., 2017; Ohrt and Turau, 2012), rendering time (Jia et al., 2018; Biørn-Hansen et al., 2017), start-up consuming time (Que et al., 2017), app starting time (Ahti et al., 2016), installation consuming time (Que et al., 2017), battery temperature (Que et al., 2017), network flow (Que et al., 2017), resources consumption (Latif et al., 2016), application speed (Heitkötter et al., 2013a)
U3	Usage patterns	User experience conventions (Ohrt and Turau, 2012), screen rotation (Dhillon and Mahmoud, 2015; Palmieri et al., 2012), device orientation (Ciman and Gaggi, 2017), accessibility features (Ohrt and Turau, 2012), frequency of use (Lachgar and Abdali, 2017), offline mode (Lachgar and Abdali, 2017; Charkaoui et al., 2015)

tions are harder to realize. 3D environments and multimedia features are particularly challenging for cross-platform frameworks (Dalmasso et al., 2013).

(U2) Performance: Poorly performing apps likely face low user acceptance. Performance comprises of aspects such as app load time, app speed for changing views and computations resulting from user interaction (*responsiveness*), perceived speed of network access, and stability. While the subjective impression is important and might differ according to individual projects' requirements, some performance aspects can be measured. This includes the start-up time, the time to awake after interruptions, and the time to shut down (Dhillon and Mahmoud, 2015). Additionally, resource utilisation can be scrutinized. This includes CPU load, memory usage, battery drain during runtime (and possibly while background services remain active), and download size (Sommer and Krusche, 2013; Ciman et al., 2014; Dalmasso et al., 2013; Ciman and Gaggi, 2015). Performance aspects need to be carefully balanced, as a pure focus on performance can negatively impact many other criteria and optimisation for other criteria might negatively affect performance (for instance by bloating an app).[2]

(U3) Usage patterns: Apps are used in typical patterns. This includes many apps that are used infrequently and some apps that are often used, although normally only for a short amount of time and often with interruptions. Users desire an "instant on" experience and continue where they left the app. Unsaved data ideally should be available even after closing the app or even after rebooting a device. Data retrieved from the Internet should stay available when temporarily loosing connectivity. Apps should align with personal workflows for information processing such as sharing with other apps or saving to persistent storage. Moreover, they should integrate with common apps for interaction with other users, such as messaging, email and social media services. Data-intensive apps, especially if they have desktop counterparts, should support synchronisation of app data across multiple devices. In particular, background synchronisation for seamless, transparent context switching is desirable. Additionally, apps should make use of platform-wide services such as a notification centre or means to store certain types of documents (such as *Apple Wallet* for boarding passes and similar documents Apple Inc., 2018b).

(U4) User authentication: User management becomes increasingly important: apps may have a purely local, single user, cloud-based accounts (e.g., enabling services such as synchronisation), or employ centralised user management with multi-device account management or role-based access rights (Kunz et al., 2014). Similarly, authentication is possible on an app-level or server-based. Apps might include session management, and they might cache login information (e.g., for a limited-functionality offline mode). Ideally, frameworks should offer several ways for user authentication, including traditional pins and passwords, gestures, based on biometric information, and voice recognition (Luca and Lindqvist, 2015). While this criterion has many implications for the app perspective (and clever features of a framework can considerably simplify the developers' job), we have put it under the user perspective to underline the importance of it for working with an app.

4. Weight profiles

In the following, we first give the rationale of weight profiles before explaining the application. We then provide notable examples.

4.1. Rationale

There are two sides to the evaluation of technological and technical criteria. First, there is the actual assessment of a phenomenon

[2] The interrelation of criteria is illustrated in Fig. 1 and discussed in Section 6.1 (p. 19 onwards).

(in this case of a cross-platform development framework) under that criterion. It should be based on facts and made as little subjective as possible. Thereby, it should also be replicable. In general, this assessment is not individual, i.e., it should be the same independent of situation, context, and assessor. Second, there is the importance of the criterion for the individual situation. This is specific to a setting and dependent on context, personal preferences, and applicability of the criterion.

To cater for this observation, criteria ought to be weighted instead of simply using their average assessment score to denote the overall assessment of a framework. This also serves the purpose of balancing different levels of technical depths of criteria, which is unavoidable given the heterogeneity of the considerations reflected in our criteria catalogue.

Despite the individuality of the weighting, developers and companies in need of a decision typically face one of a number of comparable settings. Therefore, along with our evaluation framework we propose *weight profiles*. These correspond to typical development settings; they are a kind of patterns or templates. Profiles can be used *as-is* for a quick assessment and to gain an overview. Alternatively (or successively), they can be used as the foundation for an individual assessment. Instead of needing to start from scratch, a profile provides a reasonable weighting for a typical situation, which commonly will need only some tweaking. Weight profiles are not meant to be static but to evolve with the general evolution of the mobile computing ecosystems. Thus, they also serve an important part of keeping our framework timely.

One of the experts we asked to assess our criteria catalogue (see Section 5) noted that such an "approach could help also to document thoroughly the rationale behind specific tool selections". This might be particularly important in corporate decision making, where ultimately "supervisors with or without technical experience" will decide. Additionally, weight profiles also honour a divide-and-conquer proceeding in which individual criteria are assessed by narrow-area experts, criteria categories are evaluated by experts with more general focus (compare, e.g., a backend programmer to a senior software engineer), IT managers set the weights, and finally general management makes the decisions.

4.2. Application

Each of the 33 criteria receive a weight between 1 and 7[3]. The total is 100 points, i.e., each point denotes a 1% weight. In case a criterion should be omitted in the assessment, assigning a weight of 0 is also possible.

Each evaluated criterion is assigned a score from 0 (criterion unsatisfied) to 5 (criterion fully satisfied). The overall score S of an assessment is then calculated as the weighted arithmetic mean of the criteria, i.e., as

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

with i_j, d_j, a_j, and u_j being the criteria score from the infrastructure, development, app, and user perspective, respectively, and $w_{i,j}$, $w_{d,j}$, $w_{a,j}$, and $w_{u,j}$ the corresponding weights for each criterion. We suggest rounding to two decimal digits; finer-grained scores hardly have a practical relevance. The nearer the score is to 5, the more wholesome is a framework; a score of 0 would be given to one that is entirely dysfunctional. Which ranges of scores are to be expected, sufficient, and realistic will be revisited in the discussion (Section 6) of this article.

This weighted summation is deliberately chosen to ensure simplicity and understandability for all stakeholders in the assessment process. Essential requirements such as weights being proportional to the relative value of criterion score changes are fulfilled by using equal intervals for all scores across heterogeneous criteria (Hobbs, 1980). In contrast to more advanced techniques for multi-criteria analysis such as rank-based criteria assessment or pair-wise comparisons (Barron and Barrett, 1996), the chosen approach is modular in nature such that weights and scores can be defined by different experts in the respective domain. In addition, new frameworks can easily be added to the decision process without re-evaluating all combinations. This capability is particularly suited for today's fast-changing world in which mobile development frameworks constantly appear and disappear.

4.3. Example profiles

Table 3 (following in the next section, p. 14) provides a weighting with the weights of the *smartphone* weight profile, along with the weights of five additional profiles. The smartphone profile is the most standardized profile; its weights are well backed by empirical and theoretical work as well as experience from practice.

Prior work has shown that cross-platform approaches often focus developers' need (Research2guidance, 2014). Typically, open-source approaches will be appreciated (I1 (License)). Undoubtedly, good support of the desired target platforms has a high value (I2), and a long-term feasibility will be sought (I7). In alignment with these infrastructure considerations, developers find a proper development environment (D1), an adequate preparation time (D2) and swift development progress (D12 (Pace of development)) important. For smartphone usage, a good user interface (D5 (UI Design)) is essential. Regarding the actual app, particularly proper access to device hardware (A1) and platform functionality is needed (A2). Finally, from the user perspective smartphone apps needs to provide a near-native appearance (U1 (Look and feel)) and a good runtime performance (U2). Consequently, in the smartphone weight profile $\frac{1}{3}$ of the criteria (i.e., 11) have 56% of the total weight.

This generic smartphone profile can be amended to fit with particular needs. For example, if the choice of a framework has specific strategical significance, full weight (7) could be given to I1 (License) and I7 (Long-term feasibility), and possibly a higher weight to D9 (Maintainability) and A7 (System integration). At the same time, D2 (Preparation time) and D3 (Scalability) could be assigned very low weights.

Similarly to the smartphone profile, it makes sense to provide a *tablet* profile. It could be used for apps specifically targeting tablets. Depending on the target of development, it might be used in conjunction with the smartphone profile, e.g., when apps are supposed to provide specific support to different screen sizes and orientations. The tablet profile is very similar with smaller deviation of typically only 1. However, such small deviations might tip the scales if several matured frameworks are almost equipollent with the smartphone weights.

The *entertainment* profile applicable for example to augmented/virtual reality devices has many similarities to the aforementioned ones; however, less emphasis is given to the app perspective (with exception of an app's robustness (A9) – low robustness of, e.g., games is very frustrating). Much attention is given to the user perspective to satisfy users' needs when using an app to be entertained. The notable exception is the look and feel (U1). While we routinely stress the importance of a native look and feel of an app, entertainment apps often come with a highly individual user interface. Particularly games often do not adhere to a platform's

[3] Higher weights are not only impractical but also questionable given the granularity of our criteria. With high weights for single criteria, the weight for the majority of criteria would be very low, possibly leading to a point where assessing them would make little sense since they would have no significant effect on the overall score. Our criteria are meant to offer a balanced assessment where no single criterion inherently is more important than the other.

Table 3
Comparison of frameworks and device class weight profiles for the exemplary scenario.

Smartphone Comparison								Category Weights (%)				
Criterion		Weight (%)	Web Apps	PWAs	PhoneGap	React Native	Native apps	Tablets	Entertainment	Wearables	Vehicle	Smart Home
I1	License	5	5		5	5	5	5	6	5	3	5
I2	Target Platforms	6	5	4	5	4	1	5	6	5	4	7
I3	Development Platforms	2	4		5	4	2	2	2	1	1	1
I4	Distribution Channels	2	5		3	3	4	2	3	4	3	3
I5	Monetisation	1	0		3	3	5	2	1	1	2	2
I6	Internationalisation	1	1		3	3	5	2	2	2	0	1
I7	Long-term Feasibility	5	5		5	3	4	5	3	3	5	5
D1	Development Environment	7	4		5	4	5	7	7	5	5	6
D2	Preparation Time	7	5		4	4	3	7	7	5	1	5
D3	Scalability	2	3		3	4	3	2	3	2	3	2
D4	Development Process Fit	2	3		3	3	2	2	3	1	3	2
D5	UI Design	4	3		3	2	4	4	5	5	6	2
D6	Testing	3	3		4	3	5	3	3	3	6	3
D7	Continuous Delivery	3	5		5	3	3	3	3	4	3	2
D8	Configuration Management	1	0		0	0	3	1	1	2	2	2
D9	Maintainability	2	2		4	4	2	2	2	1	3	2
D10	Extensibility	2	5		5	2	5	2	2	2	1	2
D11	Custom Code Integration	2	0		3	3	5	2	2	1	0	0
D12	Pace of Development	4	3		4	3	0	4	3	3	2	4
A1	Hardware Access	4	2		4	3	5	3	1	6	4	6
A2	Platform Functionality	5	2		4	3	5	5	3	2	2	3
A3	Connected Devices	3	0		2	2	5	2	1	5	3	7
A4	Input Heterogeneity	1	3		4	4	5	3	3	2	2	2
A5	Output Heterogeneity	1	3		4	4	5	1	1	6	3	4
A6	App Life Cycle	2	0	2	4	4	5	3	3	3	3	2
A7	System Integration	3	3		3	3	5	3	3	1	2	1
A8	Security	3	0		0	0	3	4	1	3	7	5
A9	Robustness	2	2		4	2	3	1	4	3	2	1
A10	Degree of Mobility	1	1		1	3	5	1	0	4	5	0
U1	Look and Feel	5	1	2	3	4	5	4	2	4	5	3
U2	Performance	4	2	3	2	3	5	3	6	3	3	2
U3	Usage Patterns	2	0	2	2	2	2	3	4	3	3	4
U4	User Authentication	3	0		0	0	1	2	4	0	1	4
Weighted Score			**2.87**	**2.98**	**3.59**	**3.11**	**3.73**					

interface standards at all but provide custom elements to create an *immersive* atmosphere.

Although the app-enablement of cars is just at its beginning and uncertainty exists regarding the best approach for secure and reliable platforms (Mandal et al., 2018), we propose a *vehicle* profile to illustrate a weighting that is less similar to the former three than those are to each other. Save for the long-term-feasibility (I7), less weight could be given to the infrastructure perspective. Regarding development, we consider the UI design (D5) to be particularly important, so that apps for cars can be aligned with existing infotainment systems. Moreover, testing of such apps (D6) is vital, as even in non-security critical areas app crashes and malfunctions are highly undesirable for their distraction alone. In the app perspective, we would put weight on access to hardware (A1), which would be unlike the hardware apps typically have access to. Highest importance must be given to security (A8). Particularly an app that has been given wide hardware access must not be exploitable to gain access to a car's internal functions. Additionally, the degree of mobility (A10) should ideally be very high.

Finally, due to the rise of IoT applications with interfaces to consumer usage, we deem a *smart home* profile to be utile. Even though this field is also still emerging and the weights might need adjustments in the near future, an initial assessment can be made. Due to the very high heterogeneity, adequate support of possible target platforms (I2) is essential. For the same reason, special emphasis should be put on long-term feasibility (I7) at this stage. Development aspects can get somewhat less focus but for a powerful IDE and a low preparation time, enabling a rapid start with trying out functionality – and a low penalty for changing to another framework if necessary. An app requires profound hardware access (A1), good connectivity with a wealth of other devices (A3), and a high level of security (A8).

Weight profiles are not limited to typical settings in corporate app development. One of our experts suggested that weighting

would also be useful if students work on development projects. Weighting would then be aligned with the curriculum. Thus, there could for example be a CS-123 "Mobile App Development Laboratory" profile. It would be a specialisation from the smartphone profile that discards – in this setting – superfluous criteria such as I4 (Distribution channels), I5 (Monetisation), I6 (Internationalisation), and D8 (Configuration management).

5. Evaluation study

According to the taxonomy by Majchrzak et al. (2015a), three main approaches to develop apps can be distinguished (although recently introduced frameworks such as React Native and NativeScript are blurring the lines between them):

1. app employing a runtime-environment, subdivided into (Progressive) Web Apps, hybrid apps (similar to Web Apps but wrapped in installable containers), and self-contained runtimes,
2. generative approaches, subdivided into model-driven software development and transpiling of existing apps, and
3. native development.

Obviously, only the first two qualify for cross-platform development. The following evaluation compares a heterogeneous selection of frameworks in order to demonstrate the applicability of the proposed evaluation scheme to different development approaches. In particular, native app development is contrasted to the hybrid app framework PhoneGap as well as to React Native and to (Progressive) Web Apps. PhoneGap was chosen due to its perennial prominence and popularity as leading cross-platform development tool using hybrid apps (Stack Overflow, 2018). As a relatively new approach, React Native aims to combine the advantages of native UI elements with the familiarity of JavaScript among web developers (Facebook Inc., 2018). Web Apps on the other end of the spectrum bridge the gap to traditional web development – especially with regard to the recently introduced concept of *progressive* Web Apps, which intensify the integration of web and smartphone applications.

This evaluation does *not* aim for a comprehensive survey of the investigated frameworks but rather serves as a benchmark for our evaluation criteria. Nevertheless, it can be used by researchers to scrutinize our criteria; for practitioners is can serve as a starting point. More detailed comparisons of specific frameworks are for example provided by Dhillon and Mahmoud (2015) or Hudli et al. (2015).

5.1. Method

The criteria presented in Section 3 have been assessed by several experts, both academic researchers involved in mobile app research and practitioners with experience in cross-platform development tools. Their feedback has been incorporated in the improved criteria description.

To demonstrate the applicability of our criteria catalogue and weighting scheme, we perform an evaluation study. Resulting from the recent emergence of devices, barely any (commercial or academic) framework exists that allows for cross-platform development within one or across multiple device classes regarding novel mobile devices (cf. Section 2). Our evaluation study therefore focuses on cross-platform smartphone frameworks to demonstrate the applicability of our weighted criteria approach, which can be backed with empirical and theoretical work. For better assessment, the corresponding criteria (or rather their short identifiers) are given in brackets when discussing aspects of a frameworks through Sections 5.2–5.5.

As a hands-on scenario of a cross-platform app, we consider the example of a typical business app that performs data manipulation tasks for field service workers. Salespersons need ubiquitous access to the company's information systems to support their daily work. For example, they are often away on business and frequently experience context switches in sales talks with customers or while travelling. Retrieving up-to-date information such as current inventory levels is essential for decision making. Also, performing tasks such as order placement or master data management can be accomplished efficiently. Using a mobile cross-platform app, field service workers can use a device of their choice and benefit from digitized business processes.

As elaborated in Section 4, the chosen weights presented in Table 3 (p. 14) are therefore not inalterable but adapted to this specific use case. Arguably, they represent a suitable weight profile regarding business apps for company-internal usage based on the following considerations. Previous studies have shown that cross-platform approaches are often driven by IT departments to enable efficient development for multiple target platforms (Research2guidance, 2014). From an infrastructure perspective, this means that open and extensible approaches are considered to be particularly important. Concerning long-term feasibility, the dominance of Android and iOS as main players on the mobile operating system market (Forni and van der Meulen, 2017) has created a stabilized smartphone ecosystem. Consequently, distribution channels are mostly limited to the respective platform-specific app stores, which offer a broad set of features such as limiting the deployment to a geographic region or offering multiple language versions.

App developers want to use existing standards and previous knowledge to progress quickly with the task at hand (Research2guidance, 2014). Until now, apps are commonly developed in small development teams. Hence, the organisational aspects of software engineering practices such as scalability, maintainability, and the integration in team-oriented development processes have low priority (Research2guidance, 2014). With increasing variety of devices and complexity of the apps itself, effective testing of app artefacts becomes more important when dealing with essential business activities. A targeted delivery of related apps (e.g., language-specific variants or functional variability) is, however, negligible for the given scenario. Most important, UI design using characteristic platform widgets and interaction patterns seems to be an ongoing challenge for cross-platform framework. Beyond company-internal apps, it may even become more important for "standing out from the mass of apps" (Amatya and Kurti, 2014).

On the application side, access to a broad rage of device functionalities is frequently requested by practitioners. This trend is facilitated by the convergence of input and output capabilities in the matured smartphone market. In general, apps are mostly used for entertainment and communication purposes (Lella and Lipsman, 2017); thus, business integration and security issues are specific requirements for digitisation projects allowing for the transmission of sensitive data to mobile devices.

In 2016, worldwide mobile usage has already surpassed desktop usage (StatCounter, 2016), a trend which is potentially accelerated by the plethora of upcoming wearable devices. Therefore, smartphones cannot be treated as merely displaying web content from anywhere – a platform-specific look and feel as well as performance considerations remain important topics for mobile app development. Finally, user authentication is often required for apps because a central backend or cloud environment often serves as content provider for the application or is used for additional services such as synchronisation across devices.

According to this scenario and the derived weight profile, the following subsections discuss the suitability of PWAs, Phone-

Gap, and React Native in contrast to traditional native app development. We do not justify each score separately (which would require at least $32 * 5 = 160$ sentences clumsily filled with numbers). Rather, we address particularly noteworthy parts of the evaluation, especially if scores do not intuitively make sense in the light of the frameworks and based on above considerations.

5.2. (Progressive) Web Apps

Web Apps are web sites created with web technologies – HTML5, Cascading Style Sheets (CSS), and JavaScript (JS) – that are optimized for mobile usage. They are run inside a browser, which makes them compatible with any device providing a browser that supports the features of employed versions of HTML, CSS, and JS. Consequently, Web Apps integrate well with traditional web development tooling (I1 (License), I2 (Target platforms), I3 (Development platforms)), yet app distribution cannot be controlled as no centralised app store exists (I4 (Distribution channels)). Furthermore, the web development foundation constraints its "app-like" behaviour, resulting in recurring features of apps for cloud-based services to be re-implemented manually, e.g., regarding internationalisation, payments, or user authentication (I6 (Internationalisation), D12 (Pace of development), U4 (Development process fit)). As smartphones become computationally more powerful and web development standards evolve to incorporate new (hardware) features in public APIs of browser environments, the long-term feasibility (I7) of this approach can be assured.

The recent development has led to so-called Progressive Web Apps (PWA), which need to be distinguished from "classic" Web Apps. While PWAs are also created using web technologies, their possibilities go beyond what was possible so far (Majchrzak et al., 2018). According to Google, the key characteristics to provide a better user experience are to create *reliable*, *fast*, and *engaging* applications (Google LLC, 2018h). Using *Service Workers*, PWAs can be added to a smartphone's home screen *without* the need to install them like native apps. In addition, recent standards for in-browser storage are used to load app logic and previous content from the device and provide functional applications even in situations with unavailable or unstable network connection. They might thereby close the gap between web site and native apps, providing a contender for the unifier of mobile development (Biørn-Hansen et al., 2017).

With regard to developing apps, an immense community of developers exists as standard web development skills are required; many tutorials and profound tool support are available to learn (D1 (Development environment), D2 (Preparation time)). However, this flexibility also limits the goal-oriented creation of apps. The structure of source code and UI development completely depend on the developers, for instance requiring boilerplate code. Frameworks can provide guidance by structuring the components of the app and consistently applying the concepts of PWAs (D3 (Scalability), D4 (Development process fit), D5 (UI design)). For example, the Ionic framework (Ionic, 2018) is based on the common JS library Angular and focuses on the fast creation of Web Apps though a large variety of pre-defined components for UI design and interactions. Whereas several techniques for testing JavaScript exist, desktop browsers can only inadequately emulate the characteristics of mobile devices and mobile in-browser debugging of the complete app life cycle is complicated (D6 (Testing)). On the other hand, established tool chains (for instance build tools such as Grunt or Gradle) can be used to assemble Web Apps in a continuous delivery process, especially regarding the increasing complexity of app product lines (D7 (Continuous delivery), D8 (Configuration management)).

The Ionic framework and similar libraries simplify the development of Web Apps through modular components that can be reused in multiple projects; in addition, they are extensible using third-party plug-ins (D10). The execution of native code is, however, generally unsupported within a browser environment[4] (D11 (Custom code integration)). Instead, device components can be accessed via HTML5 APIs such as Media Capture Stream and Battery Status, which are varyingly supported by mobile browsers (A1 (Hardware access), A2 (Platform functionality)) and depend on the platform vendors' willingness to support these standards (MobileHTML5, 2015). Considering input and output mechanisms, keyboard and gesture support are well established through JS events. These are based on traditional web page behaviour, thus providing only limited support for novel mechanisms such as voice-based interfaces (A4 (Input heterogeneity), A5 (Output heterogeneity), U1 (Look and feel)). Web Apps and PWAs are inherently bound to JavaScript engines, which are unavailable on most wearable devices; also, browser environments are not designed to interoperate with connected devices (A3 (Connected devices), A10 (Degree of mobility)).

As stated before, all aspects regarding system integration, security, and robustness have to be built manually based on web technology without platform-specific abstraction (A7 (System integration), A8 (Security), A9 (Robustness)). The largest advantages of PWAs are related to the usage perspective. By storing application code and previous content on the device instead of fully reloading a web site, the perceived performance is drastically improved (Google LLC, 2018g). Through service workers running in the background even after "leaving" the Web App, the application life cycle is better supported (A6). Also, the application can save user-related characteristics to the local device and instantly adapt to the users' preferences when reopening the PWA (U2 (Performance), U3 (Usage patterns)).

5.3. PhoneGap

The first stable release of PhoneGap was developed in 2009, just two years after the introduction of the first smartphones such as Apple's iPhone. Since then, it has evolved to one of the top-used cross-platform development tools (Davis, 2009; Stack Overflow, 2018). PhoneGap and its open-source foundation Apache Cordova are representatives of the hybrid app development approach. Essentially, a regular Web App is developed with HTML5 and JavaScript and subsequently wrapped in a container that uses a *Web view* component for rendering the content without browser controls. In addition, the framework provides a bridge to access native device functionality through a common JavaScript API (A1 (Hardware access), A2 (Platform functionality)). The platform-specific wrappers allow for packaging installable apps for all major smartphone platforms. These can be distributed via regular app stores and utilize their general monetisation and internationalisation features except for deeply integrated features such as in-app purchases (I1–I6).

Because of the underlying app content, most of the freedoms and restrictions of Web Apps apply likewise. The framework's structure is well documented and many tutorials are provided by the community to quickly gain momentum and start custom development from a minimal running app skeleton. Consequently, the developer can create the app using any methodology, modular structure, and web development environment (D1-D5, D12 (Pace of Development)).

[4] Current ambitions of introducing further high-level languages to browsers using binary instructions such as WebAssembly (http://webassembly.org/) open up interesting perspectives to native in-browser development, but cannot be considered a feasible alternative, yet.

Due to subtle differences between browser engines on desktop computers and smartphones, hybrid apps with embedded WebView components are more complex to test than Web Apps. Also, running a WebView interlinked with custom app code introduces new security risks Luo et al. (2011). To support the testing of intermediate app prototypes, PhoneGap offers a remote debugging interface that connects to actual devices (D6 (Testing), A8 (Security)). Furthermore, the delivery of the resulting apps is simplified through a cloud-based deployment service to build app packages without locally installed SDKs (D7 (Continuous delivery)). Through the extensive API as level of abstraction, differences across platforms are counterbalanced, which increases the overall speed of development and ensures the maintainability when platform implementations evolve (D9 (Maintainability), D12 (Pace of development). A large set of plug-ins exist to augment the PhoneGap core functionality with additional features (D10 (Extensibility)). In contrast to Web Apps, this also includes native code components that can be added to the bridge component in order to extend the JS-accessible API to further device hardware (D11 (Custom code integration)). Consequently, various input and output capabilities are supported. The *Event API* and *Device API* additionally allow for managing the overall app life cycle (A4–A6) (Apache Software Foundation, 2015; Adobe Systems Inc., 2015).

Similar to Web Apps, the approach is limited to smartphones and tablets due to the required JavaScript engine (A10 (Degree of mobility)). However, third-party plug-ins exist to connect PhoneGap apps to external devices such as Wear OS smartwatches (Gardner, 2018) (A3 (Connected devices)). With regard to the native look and feel, PhoneGap apps not automatically comply with individual platform guidelines but provide a generic mobile appearance (U1 (Look and feel)). All app functionality needs to be implemented manually (A7 (System integration), A8 (Security), A9 (Robustness), U3 (Usage patterns), U4 (User authentication)). Finally, the additional functionality through the wrapper component incurs a performance overhead (U2) as, e.g., analysed by Dhillon and Mahmoud (2015).

5.4. React Native

Traditionally, the decision of implementing apps using either a hybrid approach or native development splits the community of developers into two camps. Besides many subordinate differences, a long-lasting controversy revolves around sacrificing a convincing platform-aligned appearance for the benefit of using web technologies well-known by a large community of developers. Several papers have investigated this topic and works such as the present article aim to guide developers in choosing an approach adequate to the project-specific requirements (Mercado et al., 2016; Que et al., 2017; Angulo and Ferre, 2014). However, a variety of frameworks such as NativeScript and Flutter have recently emerged that try to bridge this chasm by creating native UI components while specifying the app with JavaScript (Progress Software Corp., 2018; Google LLC, 2018d). A popular framework, React Native, is backed by Facebook and was presented in 2015 (Occhino, 2015). Using the API of the JavaScript framework React (also called ReactJS), the whole app, including view elements, is specified using JavaScript and a template syntax called JSX. Instead of rendering the content in a browser component, the frameworks is based on a runtime approach that uses a JavaScript engine to execute business logic but transforms UI-related code into commands to the native UI elements.

React Native is distributed under the permissive MIT license (A1) and currently targets the two major platforms Android and iOS (A2). Developers can freely choose their development platform and environment that supports JavaScript (I3 (Development platform), D1 (Development environment)). As the resulting artefacts are installable app packages, capabilities and restrictions regarding distribution channels and app store features such as monetisation and internationalisation are similar to hybrid apps (I4–I6). However, due to the relative youth of the framework, long-term reliability (I7) has not yet established and the developer community is still comparably small. The current trend towards JavaScript frameworks with native UI components might change this assessment in the future. Also, being backed by Facebook reduces the risk of discontinuation before the framework has matured.

Developers experimenting with React Native benefit from an extensive documentation and profit from previous knowledge of ReactJS (D2 (Preparation time)). Due to the highly component-centred architecture of React Native, apps can be easily subdivided, which improves development and long-term maintainability (D3 (Scalability), D4 (Development process fit), D9 (Maintainability)). Generally, this structure is favourable for including third-party extensions; yet, comparatively few of them exist (D10 (Extensibility)). On the other hand, the composition of components within the reactive programming paradigm complicates UI development and testing because app interactions are hard to simulate within and no visual editor supports the custom JSX notation (D5 (Ui Design), D6 (Testing), D7 (Continuous delivery). Consequently, the advantages of using JavaScript cannot yet be fully utilized in terms of development speed, mainly because of the relatively recent introduction of the framework and supporting tools (D12 (Pace of development)).

With regard to app capabilities, 34 APIs exist to access platform functionality and hardware features (A1, A2), although some cover only individual platforms. Supporting established JavaScript input/output events allows for a decent coverage of user interaction possibilities (A4 (Input heterogeneity),A5 (Output heterogeneity)). Due to its underlying native foundation, integrating custom native code is also possible (D11 (Custom code integration)) and third-party components provide Message APIs to interact with connected devices on a case-by-case basis, e.g., Wear OS smartwatches (A3 (Connected devices)). The overall app life cycle as well as integration and security characteristics are comparable to PhoneGap apps with native components but for an additional layer of abstraction (A6 (App life cycle), A7 (System integration), A8 (Security)). Similar to other JavaScript approaches, functionality and usage patterns need to be re-implemented manually and the novelty of the approach restricts the range of predefined components (U3 (Usage patterns), U4 (User authentication), A9 (Robustness)). However, the runtime approach provides additional flexibility for future development, especially creating connectors to map the React Native APIs to non-smartphone platforms and thus support a much larger range of devices compared to web views (A10 (Degree of Mobility)).

In the resulting apps, the native UI components achieve an appearance well-adapted to the target platform. Also, a comparably low performance overhead is induced by the runtime because it can delegate performance-heavy UI computations to the native environment (U1 (Look and feel), U2 (Performance)).

5.5. Native apps

Although technically not a cross-platform approach, it makes sense to compare frameworks with the baseline of native apps to highlights the potential of cross-platform app development. In particular, the native approach does not achieve a perfect score: Despite individual platform capabilities being fully exploitable, drawbacks result from the separate development of multiple apps. Moreover, development in the native languages and using the typical environment for native development is not necessarily more efficient; in fact, even the performance of a native app

might be challenged by one that undergoes runtime optimization, as has been discussed[5] for desktop applications written in Java (Goetz, 2005). Besides, using native development might feel clumsy compared to the *convention-over-configuration* approach typical for modern frameworks (Vuksanovic and Sudarevic, 2011).

Obviously, the target platforms are limited to only one, whereas the app itself may be developed using various development platforms or technologies (I2 (Target platforms), I3 (Development platforms)). Using the freely distributed platform SDKs, a full integration of apps with the respective app store infrastructure can be achieved, including runtime interactions such as in-app payments and updates (I1 (License), I4 (Distribution channels), I5 (Monetisation), I6 (Internationalisation)). Because iOS and Android have emerged as stable duopoly of smartphone operating system vendors, long-time reliability is ensured (Forni and van der Meulen, 2017). Yet, even within such ecosystems, technological changes occur over time, e.g., introducing new programming languages such as Swift for iOS or Kotlin for Android (I7 (Long-term feasibility)).

With regard to the development perspective, knowledge of these programming languages is mandated but in general, platform vendors provide detailed documentation of the APIs and best practices to the large community of developers (D1 (Development environment), D2 (Preparation time)). In addition, platform-adapted IDEs often support the development process through visual editors for UI design (D5) or suitable tools for testing, bundling, and deploying apps (D6 (Testing), D7 (Continuous delivery)). On the other hand, this flexibility requires the developer to decide on appropriate processes to develop, scale, and maintain the app (D3 (Scalability), D4 (Development process fit), D9 (Maintainability)). For example, developers can use platform and app store features to develop multiple app versions such as regional language translations or theming, but the actual integration of provided low-level functionality needs to be performed by hand. As different programming languages and platform characteristics prohibit the reuse of code across multiple platforms, the redundant implementation of apps leads to a very inefficient pace of development (D11 (Custom code integration)).

Native apps can access all possible features of a given platform (A1–A7). Again, this flexibility is provided on a low level of platform interfaces and developers have the responsibility to adequately use the provided features. For example, platforms allow for a fine-grained control over app permissions or network connectivity, yet it is up to the developers to exploit these capabilities and react to app-external changes of context. Similarly, the app state can be monitored to integrate typical usage patterns and provide a pleasant user experience (A8 (Security), A9 (Robustness), U3 (Usage patterns), U4 (User authentication)).

Finally, native app development ensures that visual appearance and user interactions can be fully aligned with the respective platform guidelines (U1 (Look and feel)). At the same time, developers can achieve the best performance on computationally restricted mobile devices as overhead of cross-platform abstractions through frameworks or runtimes is avoided (U2 (Performance)).

5.6. Intermediate conclusions

In this section, we have demonstrated how the proposed criteria catalogue can be put into practice by performing an evaluation study. While focussing on business apps for smartphones, we compared different development paradigms for mobile apps with the help of representative and widespread frameworks. More specific, native development for multiple platforms is contrasted to purely browser-based Web Apps and enhanced Progressive Web Apps using the Ionic Framework, hybrid apps with PhoneGap/Apache Cordova, and the runtime-based approach React Native.

For the given scenario, it can be concluded that native app development supports the widest set of features but comes at the cost of multiple redundant implementations. Though simple to develop, Web Apps do not provide an adequate solution for cross-platform apps – yet Progressive Web Apps enable more app-like behaviour and significantly reduce the drawbacks of in-browser apps on supported platforms. Hybrid app frameworks such as PhoneGap on the one hand outperform the previous approaches by combining the ease of web development using technologies such as JavaScript while at the same time creating installable apps with access to hardware sensors and platform features. On the other hand, native app appearance and performance are unobtainable and critically noticed by users (Ahti et al., 2016). Finally, trending runtime-based frameworks such as React Native aim to solve the usage drawbacks of hybrid apps through native UI components while still being programmed using JavaScript. However, due to their novelty these frameworks have not yet established a large community to provide third-party libraries and equivalent coverage of functionality.

We want to stress that this evaluation is largely influenced by the scenario-specific weight profile and based on the specific frameworks' capabilities. Changing the weights would arguably lead to different recommendations. In fact, having no eminent *winner* of the evaluation underlines the benefits that situation-specific weights provide. This also has implications for research and practice: Scientific assessment of cross-platform app development is far from being complete. At the same time, practitioners are undoubtedly provided with a choice of *good* option; finding the optimal is strongly dependent on context. We recommend to do at least individual weighting, if feasible also additional assessment.

Moreover, the evaluation must be seen in the light of ongoing developments. For example, novel frameworks such as React Native might soon be on a par with long-standing frameworks when the community of developers grows and openly shares components for reuse in other projects. Also, target platforms evolve such that PWAs might be better supported in the future; thus further blurring the lines between apps and the Web. Our evaluation study should therefore rather be seen as a present-day snapshot of the mobile app development landscape instead of a *final* assessment. We by no means claim to provide a definite ranking of frameworks. On the contrary, such universal evidence does not exist but strongly relies on assumptions derived from the application domain, company guidelines, team members' experience, and project-specific requirements.

In order to provide the necessary flexibility for customised evaluations, the weight profiles can be used to tailor the comprehensive list of criteria to the respective use case as described in Section 4. Moreover, our catalogue of criteria can be applied to different classes of mobile devices and considers the varying importance of specific criteria. Consequently, this work can be used by practitioners as well as researchers for systematically selecting a suitable cross-platform development framework.

6. Discussion

Although most feedback by the experts consulted to examine (a prior iteration of) our catalogue is already incorporated in the presented set of criteria, some overarching topics are discussed in the following. In addition, we revisit literature gaps, limitations, and implications on further research.

[5] Benchmarking just-in-time compiled programs is an extremely hard endeavour (Georges et al., 2007). For native apps, research on runtime optimization is an open task.

6.1. Assessment

To validate our evaluation framework and the criteria of the catalogue, we contacted several experts from academia and practice with experience in the field of mobile app development to scrutinize the approach. More specific, open-ended questions were asked to assess the following requirements for each criterion[6]:

- **Comprehensibility**: Are descriptions clear even for people without cross-platform proficiency?
- **Unambiguousness**: Is it impossible to wrongly interpret criteria or the way how frameworks should be assessed according to them?
- **Adequacy**: Do the criteria really cover important aspects for selecting a framework?
- **Completeness**: Are the criteria collectively exhaustive and cover all relevant aspects of cross-platform framework selection?
- **Consistency**: Are descriptions free from contradictions? Are criteria mutually exclusive?
- **Verifiability**: Can the criteria realistically be used to assess cross-platform development frameworks? Are the descriptions specific enough to be operationalisable?

Additional questions dealt with the following overarching aspects:

- Are all criteria operationalisable? This includes whether we realistically describe them and whether they are not too generic.
- Do our criteria align with the "business routine" in software development? In particular, do they fit with different development methodologies (e.g., agile vs. traditional) and paradigms? And does the catalogue reflect app development practices in teams (e.g., team size, roles etc.)?
- Besides smartphones and tablets, does our catalogue sufficiently cover requirements towards app development for novel and future app-enabled devices such as smart watches, cars, smart glasses, VR devices etc.? If not, which changes or even which new criteria are needed?
- Does the catalogue constitute a good balance between practical relevance and rigorous work on the criteria?

Overall, we received very positive feedback on the criteria we deem important for evaluating cross-platform frameworks. The feedback has resulted in several clarifications to the criteria descriptions and are incorporated in the criteria catalogue presented in Section 3. The experts noted that particularly for rather generic criteria such as D12 (Pace of development) disjunction with other criteria needs to be scrutinized. Undoubtedly, full independence of criteria is unobtainable even if direct overlaps are avoided to the degree achieved by us; in the example of D12, its relationship to D2 (Preparation time) is evident. Further influences between criteria in the presented catalogue are visualized in Fig. 1.

The directed edges in Fig. 1 can be read as "having an *inherent* influence", not as "having a *constituting* influence". This means that no criterion is a superordinate or marks a kind of prerequisite for another one. Rather, some criteria share characteristics that despite all strive for cohesion are not independent in typical settings in practice.

Consider, e.g., (I2) supported target platforms. The supported platforms, and the way they are supported for example regarding different versions, cannot fully be untied from the (I7) long-term

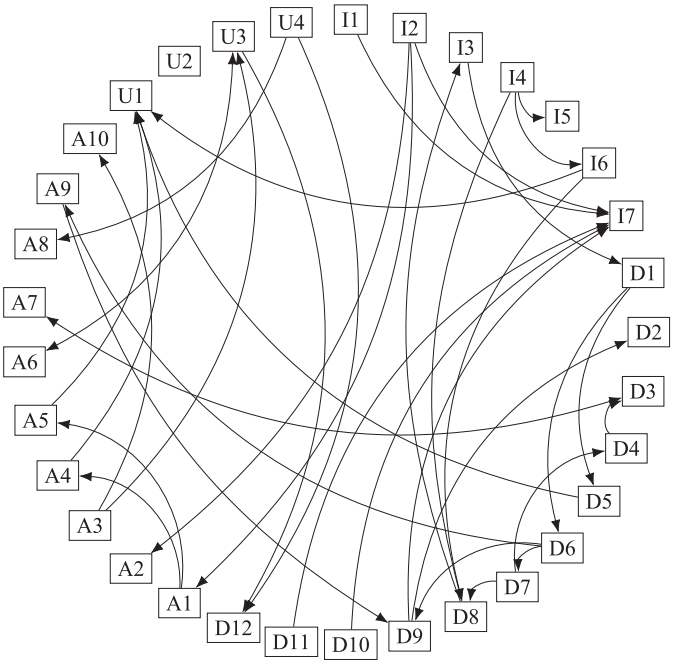

Fig. 1. Visualisation of main dependencies among evaluation criteria.

feasibility. Undoubtedly, unsatisfying platform support will hardly go along with a good long-term feasibility. However, the other way around, very good platform support does not mandate long-term feasibility, which is dependent on many other factors. Additionally, (A1) Access to Device-specific Hardware and (A2) Access to Platform-specific Functionality are influenced by I2. The worse the platform support is, the less likely it is to meet satisfying levels of hardware and platform access. Again, this is a unidirectional relationship.

Explicating such interdependencies should make it easier to assess and advance our framework. For practical usage, there should hardly be any consequences. However, with any future changes to the framework keeping the high cohesion of criteria needs to be an explicit aim.

Cross-platform app development for mobile devices extends beyond "traditional" smartphones and tablets. With novel classes of mobile devices such as smartwatches and smart TVs reaching a more widespread adoption and further ones such as connected cars, smart glasses, and augmented/virtual reality devices foreseeable (Rieger and Majchrzak, 2018), new apps might be developed for multiple platforms within such a device class or even bridging different heterogeneous devices. Therefore, our experts were further asked whether the existing set of criteria satisfies the additional requirements or what changes need to be made. Although none of the interviewed experts had previous practical experience with novel app-enabled devices from a developer perspective, to their perception the presented catalogue is apt to these new challenges.

The applicability of our criteria catalogue for real projects is an important consideration to assess whether we have managed to achieve a good balance between practical relevance and rigorous work on the criteria. In addition, the criteria catalogue needs to align with typical business routines. For example, development methodologies such as agile approaches or typical team structures such as team size and role distribution influence the ability to follow the approach presented in this work. Responses indicate that our catalogue is well applicable and can easily be tailored to a project-specific configuration. Not all criteria are equally important; therefore, the weighting approach presents a suitable

[6] We roughly follow typical assessment criteria used in requirements engineering (Sommerville, 2011, p. 94). These are handy for cases in which a comprehensive specification is desired.

solution to filter and prioritize the comprehensive set of possible criteria and adapt it to company- or project-specific requirements. One expert stated it would be helpful to have an additional checklist in order to simplify the assessment of the score for individual criteria. This was also backed by another expert who expressed concerns that some criteria require much effort for assessment (such as the robustness, A9) or are hard to verify (such as the performance, U2). We are aware of the practical benefits of a checklist; however, the fast-moving mobile domain would require a constantly updated list of current and upcoming features for a broad set of frameworks. Instead, the list of criteria relevant for cross-platform frameworks has stabilized over time and is a long-term contribution to the field of research. Moreover, we have noted when project-specific requirements should be considered.

6.2. Ongoing demand for research

The comprehensive literature search that serves as foundation for our criteria catalogue revealed five gaps that call for further research in the field of assessing mobile cross-platform frameworks.

First, the selection of criteria often appears to be chosen in an ad-hoc manner and rarely covers all four perspectives nor an exhaustive list of criteria within one specific perspective. For instance, Ribeiro and da Silva (2012) focus on the development and app perspectives but do not mention essential criteria such as the effort to set up (D2) and continuously maintain (D9) developed apps. Our weighting approach aims to guide the selection process by focusing on important criteria from the *large* catalogue. However, how to create adequate weight profiles or setting custom weights are still open questions. We have provided the tools but how to use them most effectively needs to be identified. On the one hand, the domain of (novel) mobile devices still evolves quickly and many new devices are proposed. The identified device classes might therefore change in the future and evolutionary effects of convergence (e.g., fitness devices and smartwatches share commonalities such as health tracking capabilities) or divergence (e.g., augmented and virtual reality devices might develop characteristic features when more devices appear) might be observed. On the other hand, the chosen weights in this work are convincingly argued but need to be validated through real-world projects and are open for revisions. Because of the limited availability of case studies on novel mobile devices, the overall prioritisation of criteria for different device classes needs further research (nevertheless, project-specific adaptations are always possible).

Second, a need for common benchmarks arises from the multitude of individual and incomparable evaluation studies. Researches put significant effort in the assessment of cross-platform frameworks, yet various parameters limit the comparability of their results:

- **Features**: Various software and hardware features can be considered for framework comparisons. No uniform set of features that are expected has been described.
- **Metrics**: Measuring the performance of feature usage is complex. Sensor modes (e.g., network- or hardware-based location detection), data retrieval methods (getting the last known value vs. waiting for the next update), and the amount of system- or app-specific setup instructions (event listeners, etc.) have a great impact on the outcome when measuring execution times and resource utilization. However, benchmarks with detailed specification of constituent metrics are usually omitted and thus limit reproducibility.
- **Platforms**: Commonly, the distinction is made between the two major operating systems Android and iOS but not all studies cover both (e.g., Que et al., 2017) such that results cannot be transferred. Moreover, different versions of each may qualify as distinct platforms if the internal (e.g., performance improvements by using different browser engines) or visible characteristics (e.g., the introduction of material design affecting usability) have significantly changed (El-Kassas et al., 2017).
- **Devices**: Emulated and physical devices exhibit a large variety of hardware characteristics which affect the outcome. In addition, current settings such as energy modes as well as contextual influences (e.g., network connectivity) or background tasks influence the application behaviour.
- **Framework architecture**: The underlying architecture of a framework can further complicate the comparison. For example, the performance of traditional and reactive handling of UI components is hard to assess objectively and frameworks may provide custom events for accessing sensors such as the camera which influences available metrics.

As a result, studies produce vastly different – sometimes surprising or contradicting – results, even for quantitative and seemingly objective performance measures. For instance, surprisingly slow execution times for NativeScript's video playback feature observed by Corbalan et al. (2018) cannot clearly be attributed to the device, the platform version, or the framework itself for lack of repetition. Also, studies tend to either specialize on a large set of criteria and a limited set of evaluated frameworks (e.g., Que et al., 2017) or vice versa (e.g., Umuhoza and Brambilla, 2016). Given the rapid emergence of new frameworks and the degeneration into insignificance of others, a stable set of common, repeatable benchmarks would be beneficial to enable more comparable results among different evaluation studies. For example, energy consumption is well researched for established frameworks such as PhoneGap, Sencha Touch, and Titanium (Ciman and Gaggi, 2017; Dalmasso et al., 2013) but there is no simple solution to compare these results with a measurement using a recent framework such as React Native. Automating the testing procedure to compare a large variety of devices with different platform versions as, e.g., performed by Que et al. (2017) is a worthwhile step towards this aim. However, the set of features and procedures to build such a benchmark suite need to be further researched. Also, it is not obvious whether an isolated examination of hardware or software features is sufficient or whether (and which) *app scenarios* (e.g., Biørn-Hansen et al., 2017) are better suited to assess a framework under realistic conditions.

Third, frameworks are nowadays designed to provide similar applications for different platforms and thus different users. With the advent of wearables and other app-enabled devices, app ecosystems are likely to change. For example, the same user might own multiple devices and use them in combination or subsequently depending on personal preferences (Rieger and Majchrzak, 2016). Frameworks can possibly adapt to this by reusing source code of one existing app (El-Kassas et al., 2017), applying multi-level code generation (Umuhoza and Brambilla, 2016; Rieger and Kuchen, 2019), or using other techniques related to the evolution of cross-platform software. Evaluating a framework for such a usage scenario is, however, barely considered and subject to further research. In addition, research on usability and UI design needs to scrutinize whether it is desirable that the interaction with several design classes should converge. Dependent apps for different types of devices that are unified as much as possible while keeping traits specific to (and desired for) a class would make a class-spanning cross-platform development framework even more complex than it already is. At the same time, a framework that provides such functionality would dramatically simplify developers' life.

Besides the user-centred aspects, one expert also suggested that system integration (A7) might become (even) more important in

the future, possibly even suggesting to split up the criterion. While using cloud services is essential for many apps already, the embedding of them in whole ecosystems of services and applications will likely require more attention with the expected convergence of mobile computing and IoT. When a multitude of devices provide functionality from the same app ecosystem, aspects of re-use, partitioning and distribution could also require more attention.

Finally, whereas security and data privacy issues are important especially in a business context, studies on mobile security suggest that current apps created using cross-platform tools are commonly not designed with security in mind. For example, an analysis on Apache Cordova apps by Willocx et al. (2017) revealed that best practices for the framework are mostly ignored. Integrating and evaluating the security of cross-platform apps is another challenge that has not been studied systematically and mandates further research.

6.3. Limitations

Although we deem our evaluation framework theoretically sound and usable in practice, opportunities for further research exist. Being based on existing literature and validated by experts with cross-platform experience, the criteria catalogue has a solid foundation and is expected to provide a comprehensive and rather stable set of criteria. However, the technological evolution of app-enabled devices and platforms is hard to foresee.

For example, Tesla initially announced an own SDK but reconsidered this approach due to security concerns (Lambert, 2016). New kinds of devices are designed with a focus on app-enablement – or not. Regarding platform evolution, Wear OS (Google LLC, 2018j) *might* unify development for wearables or at least consolidate different streams. Alternatively, ecosystems such as the Universal Windows Platform (Microsoft Inc., 2017) might establish themselves as integrated platforms that simplify the development for multiple devices. Also, it remains to be seen whether the success of Web technology and plethora of JavaScript frameworks for bridging the heterogeneity of devices will extend towards new device classes. So-called *instant apps* can be run without installation on smartphones (Ganapathy, 2016) and might also contribute to future changes. Yet for now, most novel mobile devices do not support JavaScript engines such as WebKit (e.g., due to hardware constraints) and require different approaches to achieve cross-platform compatibility.

Therefore, while the criteria catalogue can be expected to be relatively stable, keeping up with the technological progress will remain an inherent limitation of any work on assessing mobile computing technology.

Regarding the established weight profiles, limitations have already been mentioned in Section 6.2. More research is needed to gain empirical insights in suitable weights. Moreover, it would be utile to provide better support in criteria-specific evaluations. For examples, what would be the expected interval in which measurements are to be found for quantitative criteria? What would be sufficient expectations for qualitative ones? How would this be transferred to the score, and how could it be ensured that progress is reflected in these scores? Moreover, which ranges of scores could be expected in general, and which scores for assessment are sufficient and realistic? We have put these questions as boundaries rather than as tasks for future research for now, as answering them in a generalizable manner, aiming at the same soundness as our criteria catalogue as such, will be next to impossible.

On a more general level, it needs to be questioned whether full ecosystems similar to the current situation for smartphones will emerge for all device classes. The multitude of upcoming devices makes cross-platform development much more difficult, especially when considering the interrelations of multiple devices used in combination by the same user. A cloud-based middleware, mirroring techniques, or other “remote” approaches could solve issues such as low performance, hardware heterogeneity, and security without even relying on device-installed apps directly (Gallidabino et al., 2016; Koren and Klamma, 2016). Nonetheless, the criteria catalogue established in this work is flexible enough to deal with a wide range of technological bases which enable cross-platform capabilities.

6.4. Future work

Although we are confident that we have reached the goal of providing not *another* but the definitive framework for evaluation of cross-platform app development approaches, we will continue with our work on related topics.

One of the experts suggested that for better operationalisation of own assessments, it would be helpful to have a per-criteria checklist. Remember for instance criterion I2 (Target Platforms). The checklist could comprise Android, iOS, and Windows Mobile for smartphones. However, whether it should not contain more operating systems would be a matter of discussion, as would be how the checklist should be kept up to date. Moreover, if you think of other criteria, it can be extremely hard to propose a relatively simple list. For example, D4 (Development Process Fit) can hardly be broken down into single items that can be checked (or not). Thus, we deem research on such checklists, and in general on assessment advice (How to do it? Single choice, multiple choice, multiple select?) a target of our future work.

Ongoing research concentrates on engineering cross-platform frameworks for novel mobile devices and approaches for developing apps across multiple device classes. Besides numerous technical considerations, the prevalent conceptual challenges relate to suitable abstractions for developing apps for devices with heterogeneous input and output capabilities as well as different capabilities. Also, multi-device interactions become important when using different devices sequentially or concurrently depending on personal preferences or usage context. In the domain of data-driven business apps, model-driven approaches such as MAML (Rieger and Kuchen, 2018a; Rieger, 2018) and MD^2 (Heitkötter et al., 2013b) with high levels of abstraction are promising for extending them towards new device classes such as smartwatches. However, it is still very hard to image proper abstractions for different domains such as home automation apps for devices that fall under the umbrella of smart home technology.

In addition to the two concrete topics, we will continue with our work on building the theory of modern mobile computing. This quest will be built upon a close scrutinisation of work in practice and strive to provide rigour where fast-pace developments predominate.

7. Conclusion

In this article, we have proposed an evaluation framework for cross-platform app development approaches. Building on previous frameworks and being broken down into sound abstract criteria, it sets out to be *the* way of assessing cross-platform development frameworks for all situations in which app-enabled devices play a role.

We have shown that much literature exists yet few works have gone the lengths of providing comprehensive, holistic frameworks. Moreover, the complexity introduced by novel mobile devices is reflected in many works yet needed to be grasped and built into assessment criteria. We have provided and, in much detail, explained our criteria catalogue, which provides a synthesis of the literature. Criteria are grouped into four perspectives: infrastructure, development, app, and usage. To provide means for a tailored, customised

assessment, we have proposed the usage of weight profiles. With the help of these, a criteria-based assessment can be adapted to whatever situation is concretely met. In fact, one assessment can even be used to cater for multiple decisions.

We have demonstrated the feasibility of our framework with an evaluation study. This study has been done by applying the criteria catalogue to several app development approaches, namely (Progressive) Web Apps, PhoneGap, React Native, and native apps for comparison. The framework has been proven handy. Moreover, we have provided exemplary weight profiles. Our results have been assessed by several experts from the field of mobile computing.

We have identified much need for future research, most notably regarding the weighting, the understanding of the technological progress, and the emergence of device ecosystems. We will continue to work on solving these challenges. Eventually, we hope to be able to provide further synthesis articles such as this. The theory on modern mobile computing ought to be extended!

Acknowledgements

We would like to thank the experts who contributed to the assessment of our criteria catalogue. Detailed feedback was received from

- Henning Heitkötter, SAP SE
- Gregor Kurpiel, itemis AG
- Spyridon Xanthopoulos, University of Macedonia

Moreover, we would like to thank the three anonymous JSS reviewer who pointed out several options for improving our work.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.jss.2019.04.001.

References

Adobe Systems Inc., 2015. PhoneGap documentation. http://docs.phonegap.com.
Ahola, J., 2015. Challenges in Android Wear application development. In: Cimiano, P., Frasincar, F., Houben, G.-J., Schwabe, D. (Eds.), Engineering the Web in the Big Data Era. Springer International Publishing, Cham, pp. 601–604.
Ahti, V., Hyrynsalmi, S., Nevalainen, O., 2016. An evaluation framework for cross-platform mobile app development tools: a case analysis of adobe phonegap framework. In: ACM International Conference Proceeding Series, 1164 doi:10.1145/2983468.2983484.
Alaa, M., Zaidan, A.A., Zaidan, B.B., Talal, M., Kiah, M.L.M., 2017. A review of smart home applications based on internet of things. J. Netw. Comput. Appl. 97, 48–65. doi:10.1016/j.jnca.2017.08.017.
Alha, K., Koskinen, E., Paavilainen, J., Hamari, J., Kinnunen, J., 2014. Free-to-play games: professionals' perspectives. In: Proceedings of nordic DiGRA, 2014.
Amatya, S., Kurti, A., 2014. Cross-platform mobile development: challenges and opportunities. In: ICT Innovations 2013, 231. Springer, pp. 219–229.
Andersen-Gott, M., Ghinea, G., Bygstad, B., 2012. Why do commercial companies contribute to open source software? Int. J. Inf. Manag. 32 (2), 106–117. doi:10.1016/j.ijinfomgt.2011.10.003.
Angulo, E., Ferre, X., 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, New York, NY, USA, pp. 27:1–27:8. doi:10.1145/2662253.2662280.
Apache Software Foundation, 2015. Apache Cordova documentation. https://cordova.apache.org/docs/en/.
Apple Inc., 2018a. iOS Human Interface Guidelines. https://developer.apple.com/design/human-interface-guidelines/ios/.
Apple Inc., 2018b. Wallet - apple developer. https://developer.apple.com/wallet/.
Arp, D., Quiring, E., Wressnegger, C., Rieck, K., 2016. Bat in the Mobile: A Study on Ultrasonic Device Tracking. Computer Science Report 2016-02. Institute of System Security, Technische Universität Braunschweig.
Automatic Labs, 2018. Automatic: connect your car to your digital life. https://automatic.com/.
Barron, F.H., Barrett, B.E., 1996. Decision quality using ranked attribute weights. Manag. Sci. 42 (11), 1515–1523.
Beal, G.M., Bohlen, J.M., 1957. The diffusion process. Agricultural Experiment Station. Iowa State College.
Beck, K., 1999. Extreme Programming Explained: Embrace Change. Addison-Wesley.
Biørn-Hansen, A., Ghinea, G., 2018. Bridging the gap: investigating device-feature exposure in cross-platform development. In: Proceedings of the 51st Hawaii International Conference on System Sciences. ScholarSpace, pp. 5717–5724.
Bishop, J., 2006. Multi-platform user interface construction: achallenge for software engineering-in-the-small. In: Proceedings of the 28th International Conference on Software Engineering. ACM, pp. 751–760. doi:10.1145/1134285.1134404.
Biørn-Hansen, A., Majchrzak, T.A., Grønli, T.-M., 2017. Progressive Web Apps: The Possible Web-Native Unifier for Mobile Development. In: Proceedings of the 13th International Conference on Web Information Systems and Technologies (WEBIST). INSTICC, SciTePress, pp. 344–351. doi:10.5220/0006353703440351.
Biørn-Hansen, A., Majchrzak, T.A., Grønli, T.-M., 2018. Progressive Web Apps for the unified development of mobile applications. In: Majchrzak, T.A., Traverso, P., Krempels, K., Monfort, V. (Eds.), Revised Selected Papers WEBIST 2017. Springer, pp. 64–86.
Botella, F., Escribano, P., Peñalver, A., 2016. Selecting the best mobile framework for developing web and hybrid mobile apps. In: Proceedings of the XVII International Conference on Human Computer Interaction. ACM, New York, NY, USA, pp. 40:1–40:4. doi:10.1145/2998626.2998648.
Bouhnick, G., 2015. A List of All Operating Systems Running on Smartwatches. http://www.mobilespoon.net/2015/03/a-list-of-all-operating-systems-running.html.
Brisebois, M.A., Drummond, B., Mehta, A., Chene, M., Flannigan, M., 2017. Method and system for customizing a mobile application using a web-based interface. US Patent 9,836,446.
Carter, J., 2015. Which is the best Internet of Things platform?. http://www.techradar.com/news/-1302416.
Charkaoui, S., Adraoui, Z., Benlahmar, E.H., 2015. Cross-platform mobile development approaches. Colloquium in Information Science and Technology, CIST, 2015-January doi:10.1109/CIST.2014.7016616.
Chmielewski, J., 2013. Towards an Architecture for Future Internet Applications. In: Lecture Notes in Computer Science, 7858, pp. 214–219. doi:10.1007/978-3-642-38082-2_18.
Chun, B.-G., Curino, C., Sears, R., Shraer, A., Madden, S., Ramakrishnan, R., 2012. Mobius: unified messaging and data serving for mobile apps. In: Davies, N., Seshan, S., Zhong, L. (Eds.), Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services - MobiSys '12. ACM Press, New York, New York, USA, pp. 141–154. doi:10.1145/2307636.2307650.
Ciman, M., Gaggi, O., 2015. Measuring Energy Consumption of Cross-Platform Frameworks for Mobile Applications. In: LNBIP, 226, pp. 331–346. doi:10.1007/978-3-319-27030-2_21.
Ciman, M., Gaggi, O., 2017. An empirical analysis of energy consumption of cross-platform frameworks for mobile development. Pervasive Mob. Comput. doi:10.1016/j.pmcj.2016.10.004.
Ciman, M., Gaggi, O., Gonzo, N., 2014. Cross-platform mobile development: a study on apps with animations. In: Proc. ACM Symposium on Applied Computing doi:10.1145/2554850.2555104.
Corbalan, L., Fernandez, J., Cuitiño, A., Delia, L., Cáseres, G., Thomas, P., Pesado, P., 2018. Development frameworks for mobile devices: acomparative study about energy consumption. In: Proceedings of the 5th International Conference on Mobile Software Engineering and Systems. ACM, New York, NY, USA, pp. 191–201. doi:10.1145/3197231.3197242.
Dalmasso, I., Datta, S.K., Bonnet, C., Nikaein, N., 2013. Survey, comparison and evaluation of cross platform mobile application development tools. In: Proc. 9th IWCMC doi:10.1109/IWCMC.2013.6583580.
Dash Labs, Inc., 2018. Dash - smarter driving, every day. https://dash.by/.
Davis, L., 2009. PhoneGap: People's Choice Winner at Web 2.0 Expo Launch Pad. http://readwrite.com/2009/04/02/phone_gap.
Deindl, M., Roscher, M., Birkmeier, M., 2015. An architecture vision for an open service cloud for the smart car. Green Energy Technol. 203, 281–295. doi:10.1007/978-3-319-13194-8_15.
Delia, L., Galdamez, N., Corbalan, L., Pesado, P., Thomas, P., 2018. Approaches to mobile application development: comparative performance analysis. In: Proc. Computing Conference 2017. Institute of Electrical and Electronics Engineers Inc., pp. 652–659. doi:10.1109/SAI.2017.8252165.
Dewan, S.G., Chen, L.-d., 2014. Mobile payment adoption in the us: a cross-industry, crossplatform solution. J. Inf. Privacy Secur. 1 (2), 4–28. doi:10.1080/15536548.2005.10855765.
Dhillon, S., Mahmoud, Q.H., 2015. An evaluation framework for cross-platform mobile application development tools. Softw. Pract. Exp. 45 (10), 1331–1357. doi:10.1002/spe.2286.
Dobie, A., 2012. Why You'll Never Have the Latest Version of Android. http://www.androidcentral.com/why-you-ll-never-have-latest-version-android.
Donner, J., 2008. Research approaches to mobile use in the developing world: a review of the literature. Inf.Soc. 24 (3), 140–159.
Dorr, T., 2018. Tesla Model S JSON API. http://docs.timdorr.apiary.io.
Doud, A., 2015. How important is cross-platform wearable support?. http://pocketnow.com/2015/05/10/cross-platform-wearable-support
Durach, S., Higgen, U., Huebler, M., 2013. Smart Automotive Apps: An Approach to Context-Driven Applications. In: LNEE, 200, pp. 187–195. doi:10.1007/978-3-642-33838-0-17.
Dyck, S., Majchrzak, T.A., 2012. Identifying common characteristics in fundamental, integrated, and agile software development methodologies. In: Proc. 45th Hawaii International Conference on Systems Science (HICSS-45). IEEE Computer Society, pp. 5299–5308.
Eisenstein, J., Vanderdonckt, J., Puerta, A., 2001. Applying model-based techniques to the development of UIs for mobile computers. In: International Conference on Intelligent User Interfaces, Proceedings.
El-Kassas, W.S., Abdullah, B.A., Yousef, A.H., Wahba, A.M., 2014. ICPMD: integrated cross-platform mobile development solution. In: Proc. 9th ICCES doi:10.1109/ICCES.2014.7030977.

El-Kassas, W.S., Abdullah, B.A., Yousef, A.H., Wahba, A.M., 2017. Taxonomy of cross-platform mobile applications development approaches. Ain Shams Eng. J. 8 (2), 163–190. doi:10.1016/j.asej.2015.08.004.
Ernsting, J., Rieger, C., Wrede, F., Majchrzak, T.A., 2016. Refining a reference architecture for model-driven business apps. In: Proc. of the 12th WEBIST. SciTePress, pp. 307–316.
Facebook Inc., 2018. React native - a framework for building native apps using react. https://facebook.github.io/react-native/.
Fantacci, R., Pecorella, T., Viti, R., Carlini, C., 2014. Short paper: overcoming IoT fragmentation through standard gateway architecture. In: 2014 IEEE World Forum on Internet of Things (WF-IoT), 00, pp. 181–182. doi:10.1109/WF-IoT.2014.6803149.
Ferreira, C., Peixoto, M., Duarte, P., Torres, A., JÃºnior, M., Rocha, L., Viana, W., 2018. An evaluation of cross-platform frameworks for multimedia mobile applications development. IEEE Lat. Am. Trans. 16 (4), 1206–1212. doi:10.1109/TLA.2018.8362158.
Fitzgerald, B., 2006. The transformation of open source software. MIS Q. 30 (3), 587–598.
Forni, A.A., van der Meulen, R., 2017. Gartner Says Worldwide Sales of Smartphones Grew 9 Percent in First Quarter of 2017. https://www.gartner.com/newsroom/id/3725117.
Friese, P., 2014. Applause. https://github.com/applause/.
Gallidabino, A., Pautasso, C., Ilvonen, V., Mikkonen, T., Systä, K., Voutilainen, J.P., Taivalsaari, A., 2016. On the architecture of liquid software: Technology alternatives and design space. In: WICSA, pp. 122–127. doi:10.1109/WICSA.2016.14.
Ganapathy, S., 2016. Introducing Android Instant Apps. http://android-developers.blogspot.no/2016/05/android-instant-apps-evolving-apps.html.
Gardner, T., 2018. Android Wear Cordova Plugin. https://github.com/tgardner/cordova-androidwear.
Georges, A., Buytaert, D., Eeckhout, L., 2007. Statistically rigorous java performance evaluation. SIGPLAN Not. 42 (10), 57–76. doi:10.1145/1297105.1297033.
Gill, G.K., Kemerer, C.F., 1991. Cyclomatic complexity density and software maintenance productivity. IEEE Trans. Softw. Eng. 17 (12), 1284–1288. doi:10.1109/32.106988.
Goetz, B., 2005. Urban performance legends, revisited. https://www.ibm.com/developerworks/library/j-jtp09275/.
Google LLC, 2018a. Android developers - design for Android. https://developer.android.com/design/.
Google LLC, 2018b. Android TV. https://www.android.com/tv/.
Google LLC, 2018c. Behavior changes: all apps. https://developer.android.com/about/versions/pie/android-9.0-changes-all.
Google LLC, 2018d. Flutter - beautiful native apps in record time. https://flutter.io/.
Google LLC, 2018e. Google AdMob. https://www.google.com/admob/.
Google LLC, 2018f. J2ObjC. http://j2objc.org/.
Google LLC, 2018g. Progressive Web Apps. https://developers.google.com/web/progressive-web-apps/.
Google LLC, 2018h. Requesting Permissions at Runtime. https://developer.android.com/training/permissions/requesting.html/.
Google LLC, 2018i. Understand the Activity Lifecycle. https://developer.android.com/guide/components/activities/activity-lifecycle.
Google LLC, 2018j. Wear OS - Android developers. https://developer.android.com/wear/index.html.
HbbTV, 2018. HbbTV overview. https://www.hbbtv.org/overview/.
Heitkötter, H., Hanschke, S., Majchrzak, T.A., 2012. Comparing cross-platform development approaches for mobile applications. In: Proceedings 8th WEBIST. SciTePress, pp. 299–311.
Heitkötter, H., Hanschke, S., Majchrzak, T.A., 2013a. Evaluating Cross-platform Development Approaches for Mobile Applications. In: LNBIP, 140. Springer, pp. 120–138.
Heitkötter, H., Majchrzak, T.A., 2013. Cross-platform development of business apps with MD2. In: vom Brocke, J., Hekkala, R., Ram, S., Rossi, M. (Eds.), DESRIST. Springer, pp. 405–411. doi:10.1007/978-3-642-38827-9_29.
Heitkötter, H., Majchrzak, T.A., Kuchen, H., 2013b. Cross-platform model-driven development of mobile applications with MD2. In: Proc. SAC '13. ACM, pp. 526–533.
Heitkötter, H., Majchrzak, T.A., Ruland, B., Weber, T., 2014. Comparison of Mobile Web Frameworks. In: LNBIP. Springer, pp. 119–137.
Heitkötter, H., Kuchen, H., Majchrzak, T.A., 2015. Extending a model-driven cross-platform development approach for business apps. Sci. Comput. Program. 97, Part 1 (0), 31–36. doi:10.1016/j.scico.2013.11.013.
Hobbs, B.F., 1980. A comparison of weighting methods in power plant siting. Decis. Sci. 11 (4), 725–737.
Horsley, D., 2016. Beyond Touch: Tomorrow's Devices Will Use MEMS Ultrasound to Hear Your Gestures. https://spectrum.ieee.org/semiconductors/devices/beyond-touch-tomorrows-devices-will-use-mems-ultrasound-to-hear-your-gestures.
Hsiao, K.-L., Chen, C.-C., 2016. What drives in-app purchase intention for mobile games? An examination of perceived values and loyalty. Electron. Commer. Res. Appl. 16, 18–29.
Hudli, A., Hudli, S., Hudli, R., 2015. An evaluation framework for selection of mobile app development platform. In: Proc. 3rd MobileDeLi doi:10.1145/2846661.2846678.
Humayoun, S., Ehrhart, S., Ebert, A., 2013. Developing Mobile Apps Using Cross-Platform Frameworks: A Case Study. In: Lecture Notes in Computer Science,8004 LNCS PART 1, pp. 371–380. doi:10.1007/978-3-642-39232-0_41.
Ionic, 2018. Build amazing native apps and progressive web apps with Ionic Framework and Angular. https://ionicframework.com/.
Jacobson, I., 1999. The Unified Software Development Process. Pearson Education India.
Jakuben, B., 2013. Why developing apps for Android is fun. http://blog.teamtreehouse.com/why-developing-apps-for-android-is-fun.
Jansen, S., Bloemendal, E., 2013. Defining app stores: the role of curated marketplaces in software ecosystems. In: Herzwurm, G., Margaria, T. (Eds.), Software Business. From Physical Products to Software Services and Solutions. ICSOB 2013. In: Lecture Notes in Business Information Processing, 150. Springer, pp. 195–206. doi:10.1007/978-3-642-39336-5_19.
Jia, X., Ebone, A., Tan, Y., 2018. A performance evaluation of cross-platform mobile application development approaches. In: Proceedings of the 5th International Conference on Mobile Software Engineering and Systems. ACM, New York, NY, USA, pp. 92–93. doi:10.1145/3197231.3197252.
Jie, G., Bo, C., Shuai, Z., Junliang, C., 2015. Cross-platform Android/iOS-based smart switch control middleware in a digital home. Mob. Inf. Syst. 2015. doi:10.1155/2015/627859.
Kelley, P.G., Consolvo, S., Cranor, L.F., Jung, J., Sadeh, N., Wetherall, D., 2012. A conundrum of permissions: Installing applications on an Android smartphone. In: Blyth, J., Dietrich, S., Camp, L.J. (Eds.), Financial Cryptography and Data Security: FC 2012 Workshops, USEC and WECSR. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 68–79. doi:10.1007/978-3-642-34638-5_6.
Kim, H., Ahn, M., Hong, S., Lee, S., 2016. Wearable device control platform technology for network application development. Mob. Inf. Syst. 2016. doi:10.1155/2016/3038515.
Kline, R.B., Seffah, A., 2005. Evaluation of integrated software development environments: challenges and results from three empirical studies. Int. J. Hum. Comput. Stud. 63 (6), 607–627. doi:10.1016/j.ijhcs.2005.05.002.
Koren, I., Klamma, R., 2016. The direwolf inside you: End user development for heterogeneous web of things appliances. In: Bozzon, A., Cudre-Maroux, P., Pautasso, C. (Eds.), Proceedings 16th International Conference on Web Engineering (ICWE). Springer International Publishing, Cham, pp. 484–491. doi:10.1007/978-3-319-38791-8_35.
Kunz, M., Hummer, M., Fuchs, L., Netter, M., Pernul, G., 2014. Analyzing recent trends in enterprise identity management. In: 25th International Workshop on Database and Expert Systems Applications, pp. 273–277. doi:10.1109/DEXA.2014.62.
Lachgar, M., Abdali, A., 2017. Decision framework for mobile development methods. Int. J. Adv. Comput. Sci. Appl. 8 (2). doi:10.14569/IJACSA.2017.080215.
Lambert, F., 2016. Tesla is moving away from an SDK. http://9to5mac.com/2016/01/28/tesla-sdk-iphone-apps-mirror/.
Latif, M., Lakhrissi, Y., Nfaoui, E.H., Es-Sbai, N., 2016. Cross platform approach for mobile application development: a survey. In: International Conference on Information Technology for Organizations Development, IT4OD'16 doi:10.1109/IT4OD.2016.7479278.
Le Goaer, O., Waltham, S., 2013. Yet another DSL for cross-platforms mobile development. In: Proc. of the First Workshop on the Globalization of Domain Specific Languages. ACM, pp. 28–33.
Lella, A., Lipsman, A., 2017. The 2017 U.S. Mobile App Report. https://www.comscore.com/Insights/Presentations-and-Whitepapers/2017/The-2017-US-Mobile-App-Report.
LG Electronics, 2018. WebOS TV Developers. http://webostv.developer.lge.com/.
Li, X., Zhao, X., Iyer, L., 2018. Investigating of in-app advertising features' impact on effective clicks for different advertising formats. In: 24th Americas Conference on Information Systems, AMCIS 2018.
Lilis, G., Conus, G., Asadi, N., Kayal, M., 2017. Towards the next generation of intelligent building: an assessment study of current automation and future iot based systems with a proposal for transitional design. Sustain. Cities Soc. 28, 473–481. doi:10.1016/j.scs.2016.08.019.
Linux Foundation, 2018. Tizen. https://www.tizen.org.
Liu, R., Lin, F.X., 2016. Understanding the characteristics of android wear os. In: Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services. ACM, New York, NY, USA, pp. 151–164. doi:10.1145/2906388.2906398.
Luca, A.D., Lindqvist, J., 2015. Is secure and usable smartphone authentication asking too much? Computer 48 (5), 64–68. doi:10.1109/MC.2015.134.
Luo, T., Hao, H., Du, W., Wang, Y., Yin, H., 2011. Attacks on webview in the android system. In: Proceedings of the 27th Annual Computer Security Applications Conference, pp. 343–352.
Majchrzak, T.A., Biørn-Hansen, A., Grønli, T.-M., 2018. Progressive web apps: the definite approach to cross-platform development? In: Proceedings 51th Hawaii International Conference on Systems Science (HICSS-51). AIS Electronic Library (AISeL).
Majchrzak, T.A., Ernsting, J., Kuchen, H., 2015. Achieving business practicability of model-driven cross-platform apps. Open J. Inf. Syst. 2 (2), 3–14.
Majchrzak, T.A., Heitkötter, H., 2013. Status Quo and Best Practices of App Development in Regional Companies. In: Lecture Notes in Computer Science. Springer, pp. 189–206.
Majchrzak, T.A., Schulte, M., 2015. Context-dependent testing of applications for mobile devices. Open J. Web Technol. 2 (1), 27–39.
Majchrzak, T.A., Wolf, S., Abbassi, P., 2015. Comparing the Capabilities of Mobile Platforms for Business App Development. In: LNBIP. Springer, pp. 70–88. doi:10.1007/978-3-319-24366-5_6.

Mandal, A.K., Cortesi, A., Ferrara, P., Panarotto, F., Spoto, F., 2018. Vulnerability analysis of android auto infotainment apps. In: 15th ACM International Conference on Computing Frontiers. ACM, pp. 183–190. doi:10.1145/3203217.3203278.

Mercado, I.T., Munaiah, N., Meneely, A., 2016. The impact of cross-platform development approaches for mobile applications from the user's perspective. In: WAMA 2016 - Proceedings of the International Workshop on App Market Analytics, co-located with FSE 2016 doi:10.1145/2993259.2993268.

Messaoudi, F., Simon, G., Ksentini, A., 2015. Dissecting games engines: the case of unity3d. In: Proceedings of the 2015 International Workshop on Network and Systems Support for Games. IEEE Press, Piscataway, NJ, USA, pp. 4:1–4:6.

Meuter, M.L., Ostrom, A.L., Roundtree, R.I., Bitner, M.J., 2000. Self-service technologies: understanding customer satisfaction with technology-based service encounters. J. Mark. 64 (3), 50–64.

Microsoft Corp., 2018. Microsoft band. https://www.microsoft.com/en-us/band/.

Microsoft Inc., 2017. Develop apps for the universal windows platform (UWP). https://docs.microsoft.com/en-us/visualstudio/cross-platform/develop-apps-for-the-universal-windows-platform-uwp.

Mir, M., Dangerfield, B., 2013. Propagating a digital divide: diffusion of mobile telecommunication services in Pakistan. Technol. Forecast. Soc. Change 80 (5), 992–1001. doi:10.1016/j.techfore.2012.08.006.

MobileHTML5, 2015. Mobile HTML5 compatibility. http://mobilehtml5.org/.

Moj.io Inc., 2018. Mojio - connected car platform. https://www.moj.io/.

Nanjappan, V., Liang, H.-N., Lau, K., Choi, J., Kim, K.K., 2017. Clothing-based wearable sensors for unobtrusive interactions with mobile devices. In: 2017 International SoC Design Conference (ISOCC). IEEE, pp. 139–140. doi:10.1109/ISOCC.2017.8368837.

Neate, T., Jones, M., Evans, M., 2017. Cross-device media: a review of second screening and multi-device television. Pers. Ubiquitous Comput. 21 (2), 391–405. doi:10.1007/s00779-017-1016-2.

Ng, Y.Y., Zhou, H., Ji, Z., Luo, H., Dong, Y., 2014. Which Android app store can be trusted in china? In: Computer Software and Applications Conference (COMPSAC), 2014 IEEE 38th Annual. IEEE, pp. 509–518.

Noreikis, M., Butkus, P., Nurminen, J.K., 2014. In-vehicle application for multimodal route planning and analysis. In: Proc. IEEE 3rd CloudNet doi:10.1109/CloudNet.2014.6969020.

Occhino, T., 2015. React Native: Bringing Modern Web Techniques to Mobile. https://code.facebook.com/posts/1014532261909640/react-native-bringing-modern-web-techniques-to-mobile/.

Ohrt, J., Turau, V., 2012. Cross-platform development tools for smartphone applications. IEEE Comput. 45 (9), 72–79. doi:10.1109/MC.2012.121.

Palmieri, M., Singh, I., Cicchetti, A., 2012. Comparison of cross-platform mobile development tools. In: Proc. 16th ICIN. IEEE, pp. 179–186. doi:10.1109/ICIN.2012.6376023.

Parker, D.B., 1998. Fighting Computer Crime: A New Framework for Protecting Information. John Wiley & Sons, Inc, New York, NY, USA.

Pénard, T., Poussing, N., Zomo Yebe, G., Ella, N., 2012. Comparing the determinants of internet and cell phone use in africa: evidence from gabon. Communic. Strateg. (86) 65–83.

Perakakis, E., Ghinea, G., 2015. HTML5 technologies for effective cross-platform interactive/smart TV advertising. IEEE Trans. HMS 45 (4), 534–539. doi:10.1109/THMS.2015.2401975.

Perakakis, E., Ghinea, G., 2015. A proposed model for cross-platform web 3D applications on Smart TV systems. In: Proc. 20th Web3D doi:10.1145/2775292.2778303.

Progress Software Corp., 2018. How NativeScript works. https://docs.nativescript.org/core-concepts/technical-overview.

QSM, 2009. Function Point Languages Table: Version 5.0. http://www.qsm.com/resources/function-point-languages-table.

Quaresma, M., Gonçalves, R., 2014. Usability Analysis of Smartphone Applications for Drivers. In: Lecture Notes in Computer Science, 8517, pp. 352–362. doi:10.1007/978-3-319-07668-3_34.

Que, P., Guo, X., Zhu, M., 2017. A comprehensive comparison between hybrid and native app paradigms. In: G.S., T. (Ed.), Proceedings - 2016 8th International Conference on Computational Intelligence and Communication Networks, CICN 2016. IEEE, pp. 611–614. doi:10.1109/CICN.2016.125.

Rahul Raj, C.P., Tolety, S.B., 2012. A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. In: 2012 Annual IEEE India Conference (INDICON), pp. 625–629. doi:10.1109/INDCON.2012.6420693.

Rawassizadeh, R., Price, B.A., Petre, M., 2014. Wearables: has the age of smartwatches finally arrived? Commun. ACM 58 (1), 45–47. doi:10.1145/2629633.

Research2guidance, 2014. Cross-platform tool benchmarking 2014. http://research2guidance.com/product/cross-platform-tool-benchmarking-2014/.

Revest, F., 2018. AsteroidOS. http://asteroidos.org/.

Ribeiro, A., da Silva, A.R., 2012. Survey on cross-platforms and languages for mobile apps. In: Eighth International Conference on the Quality of Information and Communications Technology, pp. 255–260. doi:10.1109/QUATIC.2012.56.

Rieger, C., 2018. Evaluating a graphical model-driven approach to codeless business app development. In: Hawaii International Conference on System Sciences (HICSS-51), pp. 5725–5734.

Rieger, C., Kuchen, H., 2018. A process-oriented modeling approach for graphical development of mobile business apps. Comput. Lang. Syst. Struct. 53, 43–58. doi:10.1016/j.cl.2018.01.001.

Rieger, C., Kuchen, H., 2018. Towards model-driven business apps for wearables. In: Younas, M., Awan, I., Ghinea, G., Catalan Cid, M. (Eds.), Mobile Web and Intelligent Information Systems. Springer International Publishing, Cham, pp. 3–17. doi:10.1007/978-3-319-97163-6_1.

Rieger, C., Kuchen, H., 2019. A model-driven cross-platform app development process for heterogeneous device classes. In: Hawaii International Conference on System Sciences (HICSS-52). Maui, Hawaii, USA, pp. 7431–7440.

Rieger, C., Majchrzak, T.A., 2016. Weighted evaluation framework for cross-platform app development approaches. In: Wrycza, S. (Ed.), Information Systems: Development, Research, Applications, Education: 9th SIGSAND/PLAIS EuroSymposium. Springer, pp. 18–39. doi:10.1007/978-3-319-46642-2_2.

Rieger, C., Majchrzak, T.A., 2018. A taxonomy for app-enabled devices: Mastering the mobile device jungle. In: Majchrzak, T.A., Traverso, P., Krempels, K.-H., Monfort, V. (Eds.), Web Information Systems and Technologies. Springer International Publishing, Cham, pp. 202–220.

Rodriguez Garzon, S., Poguntke, M., 2012. The Personal Adaptive in-car HMI: Integration of External Applications for Personalized Use. In: LNCS, 7138, pp. 35–46. doi:10.1007/978-3-642-28509-7_5.

Royce, W.W., 1970. The development of large software systems. In: Proc. IEEE WESCON 1970. IEEE CS, pp. 328–338.

Russell, A., 2015. Progressive Web Apps: Escaping tabs without losing our soul. https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/.

Ryu, D., Krompiec, P.K., Lee, E., Park, K., 2014. A serious game design for english education on Smart TV platform. Pro. ISCE doi:10.1109/ISCE.2014.6884479.

Samsung, 2014. Samsung: Let's talk about the design of the Galaxy Note Edge. https://news.samsung.com/global/lets-talk-about-the-design-of-the-galaxy-note-edge.

Samsung, 2018. TOAST - Samsung developers. https://developer.samsung.com/tv/develop/extension-libraries/toast/.

Sansour, R.N., Kafri, N., Sabha, M.N., 2014. A survey on mobile multimedia application development frameworks. In: Proc. ICMCS doi:10.1109/ICMCS.2014.6911207.

Schilit, B., Adams, N., Want, R., 1994. Context-aware computing applications. In: Proc. of the 1994 1st WMCSA. IEEE CS, pp. 85–90.

Schuermans, S., Vakulenko, M., 2014. Apps for connected cars? Your mileage may vary. http://trendscan.info/blog/2014/04/28/apps-for-connected-cars-your-mileage-may-vary/.

Seyed, T., Azazi, A., Chan, E., Wang, Y., Maurer, F., 2015. SoD-toolkit: a toolkit for interactively prototyping and developing multi-sensor, multi-device environments. In: Proceedings of the 2015 ACM International Conference on Interactive Tabletops and Surfaces, ITS 2015 doi:10.1145/2817721.2817750.

Shepperd, M., 1988. A critique of cyclomatic complexity as a software metric. Softw. Eng. J. 3, 30–36. doi:10.1049/sej.1988.0003. (6).

Silva, L.C.D., Morikawa, C., Petra, I.M., 2012. State of the art of smart homes. Eng. Appl. Artif. Intell. 25 (7), 1313–1321. doi:10.1016/j.engappai.2012.05.002.

Sohn, H.-J., Lee, M.-G., Seong, B.-M., Kim, J.-B., 2015. Quality evaluation criteria based on open source mobile HTML5 UI framework for development of cross-platform. IJSEIA 9 (6), 1–12. doi:10.14257/ijseia.2015.9.6.01.

Sommer, A., Krusche, S., 2013. Evaluation of cross-platform frameworks for mobile applications. LNI P-215.

Sommerville, I., 2011. Software Engineering, ninth ed. Pearson.

Stack Overflow, 2018. Developer survey results. https://insights.stackoverflow.com/survey/2018.

StatCounter, 2016. Mobile and tablet internet usage exceeds desktop for first time worldwide. http://gs.statcounter.com/press/mobile-and-tablet-internet-usage-exceeds-desktop-for-first-time-worldwide.

Statista Inc., 2018. Statista. http://www.statista.com/.

Tang, A.K.Y., 2016. Mobile app monetization: app business models in the digital era. Int. J. Innov. Manag.Technol. 7 (5), 224.

Umuhoza, E., Brambilla, M., 2016. Model driven development approaches for mobile applications: a survey. In: Younas, M., Awan, I., Kryvinska, N., Strauss, C., van Thanh, D. (Eds.), Mobile Web and Intelligent Information Systems: 13th International Conference. Springer International, pp. 93–107. doi:10.1007/978-3-319-44215-0_8.

Unity Technologies, 2018. Unity game engine. https://unity3d.com/de.

Vilček, T., Jakopec, T., 2017. Comparative analysis of tools for development of native and hybrid mobile applications. In: International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 1516–1521. doi:10.23919/MIPRO.2017.7973662.

Vitols, G., Smits, I., Bogdanov, O., 2013. Cross-platform solution for development of mobile applications. In: ICEIS 2013 - Proceedings of the 15th International Conference on Enterprise Information Systems, 2, pp. 273–277.

Vuksanovic, I.P., Sudarevic, B., 2011. Use of web application frameworks in the development of small applications. In: 2011 Proceedings of the 34th International Convention MIPRO, pp. 458–462.

Wagner, L., 2017. Turbocharging the web. IEEE Spectr. 54 (12), 48–53. doi:10.1109/MSPEC.2017.8118483.

Wasserman, A.I., 2010. Software engineering issues for mobile application development. In: Roman, G.-C., Sullivan, K. (Eds.), Proc. FoSER '10, p. 397. doi:10.1145/1882362.1882443.

Watanabe, T., Akiyama, M., Kanei, F., Shioji, E., Takata, Y., Sun, B., Ishi, Y., Shibahara, T., Yagi, T., Mori, T., 2017. Understanding the origins of mobile app vulnerabilities: a large-scale measurement study of free and paid apps. In: Proceedings of the 14th International Conference on Mining Software Repositories. IEEE Press, Piscataway, NJ, USA, pp. 14–24. doi:10.1109/MSR.2017.23.

Willocx, M., Vossaert, J., Naessens, V., 2015. A quantitative assessment of performance in mobile app development tools. In: Proc. 3rd Int. Conf. on Mobile Services doi:10.1109/MobServ.2015.68.

Willocx, M., Vossaert, J., Naessens, V., 2017. Security analysis of cordova applications in google play. In: 12th International Conference on Availability, Reliability and Security. ACM, pp. 46:1–46:7. doi:10.1145/3098954.3103162.

Wolf, F., 2013. Will vehicles go the mobile way? Merits and challenges arising by car-apps. In: Proc. 10th ICINCO, 2.

Xanthopoulos, S., Xinogalos, S., 2013. A comparative analysis of cross-platform development approaches for mobile applications. In: Proc. 6th BCI. ACM, pp. 213–220. doi:10.1145/2490257.2490292.

XBMC Foundation, 2018. Third-party forks and derivatives. http://kodi.wiki/view/Third-party_forks_and_derivatives.

Xie, J., 2012. Research on key technologies base Unity3D game engine. In: 2012 7th International Conference on Computer Science Education (ICCSE), pp. 695–699. doi:10.1109/ICCSE.2012.6295169.

Zhang, J., Chen, C., Ma, J., He, N., Ren, Y., 2011. Usink: smartphone-based moible sink for wireless sensor networks. In: Proc. CCNC'2011 doi:10.1109/CCNC.2011.5766639.

Zhu, K.X., Zhou, Z.Z., 2012. Research note-lock-in strategy in software competition: open-source software vs. proprietary software. Inf. Syst. Res. 23 (2), 536–545.

Christoph Rieger is a postgraduate researcher and Ph.D. candidate at the Department of Information Systems, University of Münster, where he also received his B.Sc. and M.Sc. degrees in Information Systems. His research focuses on Software Engineering topics, in particular model-driven software development and domain-specific languages to ease the transformation from abstract domain-oriented models into software artefacts. Application areas include technical notations such as for distributed high-performance computing as well as the domain of mobile apps for including business experts in the development process. Christoph is a member of the ACM and the Gesellschaft für Informatik e.V.

Tim A. Majchrzak is an associate professor at the Department of Information Systems at the University of Agder (UiA) in Kristiansand, Norway. He also is a member of the Centre for Integrated Emergency Management (CIEM) at UiA. Tim received B.Sc. and M.Sc. degrees in Information Systems and a Ph.D. in economics (Dr. rer. pol.) from the University of Münster, Germany. His research comprises both technical and organizational aspects of Software Engineering, typically in the context of Mobile Computing. He has also published work on diverse interdisciplinary Information Systems topics, most notably targeting Crisis Prevention and Management. Tim's research projects typically have an interface to industry and society. He is a senior member of the IEEE and the IEEE Computer Society, and a member of the Gesellschaft für Informatik e.V.