

Weighted Evaluation Framework for Cross-Platform App Development Approaches

Christoph Rieger¹ and Tim A. Majchrzak²(✉)

¹ ERCIS, University of Münster, Münster, Germany
`christoph.rieger@ercis.de`

² ERCIS, University of Agder, Kristiansand, Norway
`tima@ercis.de`

Abstract. Cross-platform app development is very challenging, although only two platforms with significant market share (iOS and Android) remain. While device fragmentation – multiple, only partly compatible versions of a platform – has been complicating matters already, the need to target different device classes is a new emergence. Smartphones and tablets are relatively similar but app-enabled devices such as TVs and even cars typically have differing capabilities. To facilitate usage of cross-platform app development approaches, we present work on an evaluation framework. Our framework provides a set of up-to-date evaluation criteria. Unlike prior work on this topic, it offers weighted assessment to cater for varieties in targeted device classes. Besides motivating and explaining the evaluation criteria, we present an exemplary application for one development approach and, as benchmarks, for native apps and Webapps. Our findings suggest that the proliferation of app-enabled devices amplifies the need for improved development support.

Keywords: App · Mobile computing · Mobile application · Cross-platform · Multi-platform · Evaluation

1 Introduction

Only two platforms for smartphone and tablet devices with significant market share remain [67]. Even developing applications *only* for Apple’s iOS and Google’s Android is challenging (cf. e.g. [41]). Essentially, apps need to be realized separately for both, doubling the effort and prolonging the time-to-market [29]. Moreover, device fragmentation – the parallel usage of several versions and possibly vendor-specific additions – complicates development, particularly for Android [17]. Cross-platform development frameworks promise to relieve developers from the hardships of considering idiosyncrasies of several platforms and versions by providing uniform development interfaces [28].

As an additional challenge for developers, an increasing number of devices is *app-enabled*. Arguably, most apps in today’s sense target the smartphone but

soon they will be routinely used on a much wider variety of platforms. Modern entertainment technology such as TVs, BluRay players and game consoles are capable of *running* apps. Cars are seen as a major target of tomorrow’s apps [66]. Wearable devices such as smartwatches and augmented reality glasses introduce novel usage scenarios [39]. Although it can be rightfully doubted that a fridge will be the main unit to install new apps on, it is likely that with the advent of the Internet of Things (IoT) many more devices will run apps. In consequence, catering for *all* device-specific particularities will become much harder.

Extending the already well-understood general requirements for cross-platform app development, we suggest taking into account the multitude of potential devices. Heitkötter, Hanschke and Majchrzak have proposed an evaluation framework for cross-platform approaches in 2012 [27]. The extension [28] is still very useful and routinely cited in current papers on cross-platform development. However, even though the set of criteria they proposed is based on a thorough foundation, the rapid proliferation of the field mandates an overhaul. In addition, the former focus on smartphone and tablet devices ignores the plethora of novel devices reaching the consumer market.

We build on the existing work on cross-platform technology. In particular, we use the existing set of criteria [28] as the foundation to provide an extended, revised catalogue of them. This allows matching the criteria with the status quo of mobile computing. An approach that is best suited for smartphone apps might fail if apps also target entertainment systems. However, selecting a “catch-all” solution might be inferior to a specific one if only handheld devices should be supported. Therefore, we not only extend criteria but also embed them in a framework that includes a *weighting scheme*. The assessment of platforms can thereby be employed to make a per-scenario choice.

This paper makes several contributions. Firstly, it provides an evaluation framework for cross-platform development frameworks in the domain of mobile consumer devices. Evaluation criteria are explained in detail and the rationale for employing them is highlighted. Secondly, we provide the means to use our framework in an individual – particularly in a device-class specific – way by proposing the integration of balanced weights. Thirdly, we demonstrate the feasibility of our work with an exemplary evaluation. Accordingly, in Sect. 2 we discuss related work and in Sect. 3 we give the necessary background. Based on this foundation, Sect. 4 provides our catalogue of evaluation criteria. The exemplary use of the criteria for evaluation follows in Sect. 5. The findings are then discussed in Sect. 6, which leads to a conclusion in Sect. 7.

2 Related Work

Since cross-platform development of apps has been a topic for a few years now, there is plenty of scientific work on the topic in general. However, most papers tackle single frameworks or have an experimental nature. Consequently, there are relatively few papers that provide an overview, and even less than offer an evaluation. A comprehensive summary of related literature regarding covered

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

Paper	Year	Evaluated tools	Main categories (number of criteria)	Focal areas
[47]	2012	RhoMobile, PhoneGap, DragonRad, MoSync	Platform compatibility (2), development features (4), general features (4), device APIs (17)	Qualitative tool comparison
[13]	2013	PhoneGap, jQuery Mobile, Sencha Touch, Titanium	Platform support, rich user interface, back-end communication, security, app extensions, power consumption, device features, open-source	Performance evaluation (memory, CPU, power consumption)
[61]	2013	Titanium, Rhodes, PhoneGap, Sencha Touch	Functionality (8), usability (6), developer support (4), reliability/performance (4), deployment (8)	Criteria definition and qualitative tool comparison
[68]	2013	<i>none</i> (cross-platform approaches in general)	Distribution, programming languages, hardware & data access, user interface, perceived performance	Criteria definition
[11]	2014	MoSync, Titanium, jQuery Mobile, PhoneGap	License, community, API, tutorials, complexity, IDE, devices, GUI, knowledge	Qualitative tool comparison and apps with animations
[10]	2015	PhoneGap, Titanium	Battery consumption, device resource usage	Evaluation of battery consumption
[16]	2015	PhoneGap, Titanium, Adobe Air, MoSync	Tool capabilities (9), performance (5), developer support (2)	Performance benchmarks and development experience discussion
[32]	2015	AngularJS, jQuery Mobile, HTML5/JS, RhoMobile, PhoneGap, Sencha Touch	Platform support (4), development support (7), deployment factors (6)	Criteria definition and qualitative tool comparison

tools, criteria and focal areas of comparison is given in Table 1. In the following, we only comment on particularly notable details.

The papers by Heitkötter et al. [27, 28] have been used as basis for further research on apps. Examples include the definition of quality criteria for HTML5 frameworks [60], quantitative performance evaluations [65], and the creation of the cross-platform development frameworks ICPMD [21] and MD² [31].

Early papers have typically taken into account few criteria only (if at all [7, 57]) – e.g. only seven [47], and only from a developer’s perspective [11]. Few works

take a rather comprehensive approach. For example, Ohrt and Turau [46] have analysed nine tools with taking a developer focus and assessing user expectations. They e.g. have had a look at programming language, compilation without SDK, code completion, GUI designer, debugger, emulator, and extensibility with native code, as well as launch time, app package size, and memory usage.

Many papers focus on particular aspects, e.g. animations [11], performance [13], and energy consumption [10]. Nonetheless, most authors at least provide criteria grouped into common categories [32, 46, 61, 68]. One problem typically found is a shortage of explanations (c.f. e.g. [32]).

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

Several papers address *smart TVs*. Typically, a combination of HTML5 and JavaScript is proposed to enable cross-platform development. Sub-topics are interactive ads [48], serious games [55], and 3D content [49]. Work on *wearables* is more scarce [34]. Some authors have proposed middleware approaches to achieve a broader device-span [9], in one case even on the hardware layer [70]. Despite much blurriness, *smart homes* could be a future area of cross-platform research [33].

Contrasting the hype around multimedia novelties for cars, few scientific papers tackle *in-vehicle apps*. Current discussions revolve around general challenges and potential applications [66], the integration of non-automotive applications into the automotive environment [54], and usability [51]. A few papers provide experimental implementations of novel concepts such as a route planning app for head-up (HUD) displays [45], an *Open Service Cloud* for cars [15], and “remote” human machine interfaces (HMI) [20]. While these papers help to understand the possibilities of cars as a potential target of apps, they are far away from actually discussing cross-platform challenges and chances.

3 Background

As a prerequisite for a differentiated evaluation of mobile platforms, we need to categorize the variety of devices. From our understanding, *mobile consumer devices* are designed to be used in absence of *stationary* workstation hardware by non-business users. While formerly it was possible to categorise by operation system, this is not feasible anymore: e.g. Windows 10 spans device classes. As no such classification exists in scientific literature, we propose a simple subdivision.

This list is not meant as a *proven* categorisation but as a working scheme for this paper. Thus, we refrain from an elaborated explanation. Within each of these device classes, a multitude of devices based on different platforms has emerged. Whereas Android and iOS have divided most of the market share of

Traditional general-purpose devices	Novel mobile devices
Smartphone	Smart TVs and entertainment devices ^a
Tablets, including hybrids such as netbooks and (so called) ultrabooks	Wearables
	Smart watches, e.g. iWatch, Pebble, Samsung Gear
	Sensing devices, e.g. fitness trackers, GPS watches
	Smart glasses for augmented reality, e.g. Google glass, MS Hololens
	Vehicles, e.g. from BMW, Tesla, Ford
	Smart home applications ^b

^a While such devices impose themselves as being included in a categorisation as such, they arguably are not *mobile* in the sense of all other devices named here.

^b This field is still very fuzzy but rapid proliferation mandates naming it here already.

smartphone platforms amongst themselves [67], competition among the novel mobile device platforms is high and no clear winners are foreseeable. A short overview of this field is provided next.

App-enabled smart TVs are already present in 35 % of U.S. households [62] and two approaches of development exist: middleware and frameworks. Over 90 % of connected TVs sold in Germany support the HTML5-based HbbTV standard that has evolved from previous approaches such as CE-HTML and Open IPTV [26, 62]. In addition, many individual frameworks emerged, for example the open-source media centre Kodi/XBMC with various forks, Android TV, Tizen OS for TV, and webOS [2, 38, 63, 69].

With smartwatches, Google and Apple again compete for dominance with their respective Android Wear and watchOS platforms. Pebble OS, Tizen OS, and webOS are further players in this domain [6]. Whereas several vendors open-sourced their operating system, few vendor-agnostic platforms exist such as AsteroidOS [53]. Other wearable devices such as fitness trackers usually ship with proprietary platforms, e.g. Microsoft Band and Firefox OS for Wearables. Those devices often support pairing with smartphones of multiple platforms; however app development is still limited. Vendors such as Fitbit and Garmin do not even produce devices with modifiable operating system [6].

Concerning the upcoming connected cars, there are four approaches for developing in-vehicle apps [59]. 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 allow access and control of features such as door locks through a remote API. Examples include General Motors, Airbiquity, and an unofficial API for Tesla cars [18]. Third, platforms including Apple CarPlay and the MirrorLink alliance use screen mirroring of apps running on the smartphone and displayed on the car's screen [20]. This approach honours security concerns by car manufacturers. Fourth, Dash Labs, Mojio, Carvoyant, or Automatic connect to the on-board diagnostics port to

interact with the car. Although this requires a Bluetooth dongle as additional hardware, many cars can be supported that are not designed to be app-enabled in the first place. In addition to this variety of approaches, distribution of apps is a challenge because of the underlying fight for dominance between car manufacturers “owning” the car platform [59].

This overview of technologies shows similar characteristics of fragmentation as the smartphone market several years ago [28]. However, few cross-platform approaches currently exist in the domain of novel mobile consumer devices. Interestingly from a cross-platform perspective, many smart TV platforms natively support app development using Web technologies such as HTML5 and JavaScript, thus being well-suited for cross-platform approaches. Some platforms such as Android and Tizen have branches that run on multiple devices from TVs to smartwatches, potentially allowing for a 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 [56].

The other way around, smartwatches can be paired with more than one platform [19]. Such apps that act as a (smartphone) device extension are current practice and thus cross-platform development approaches must consider and support each combination of host and watch platform. However, some smartwatch platforms are announcing stand-alone capabilities [25].

Several platforms claim to be the adequate open platforms for smart home and IoT applications. Qualcomm’s AllJoyn, Intel’s IoTivity, Apple HomeKit, and Google Brillo are the most important players that try to establish their middleware as comprehensive solution [8].

Finally, for in-vehicle apps, no widespread cross-platform frameworks exist due to the novelty of devices and a lack of platform accessibility. Potentially, a middleware approach [15] might be an option to provide an open ground for developers and at the same time guarantee security.

4 Criteria

In the following, we propose our categorisation framework. We start by discussing methodological considerations before explaining the criteria.

4.1 General Considerations

As argued in Sect. 2, we have been inspired by existing evaluation frameworks. Facilitating the requirements arising from the broad scope intended for our framework and catering for the progress in the field in the meantime, we propose numerous extensions and revisions. Most notably, we do not distinguish criteria by two perspectives (*infrastructure* and *development* originally [28]) but by four.

The *infrastructure* perspective describes the general background and prospect of a cross-platform development approach. The *development* perspective takes

into account aspects of using an approach for carrying out the actual programming activities. In addition to these, we introduce the *app* perspective and the *usage* perspective. The former offers an assessment of the capabilities of apps that can be realized with a given approach. This not only leads to more clarity with regard to the distinction of actual development activities and the outcome of development but also has multi-device class support in mind. While development might not differ much for different classes of devices, the capabilities of an app might vary significantly. The usage perspective considers usability, ergonomic, and performance aspects that are essential factors for user acceptance.

The categorisation into four perspective allows focussing on relevant aspects for particular needs. These needs might arise from the targeted device class(es) but may as well come from other sources. An example could be a specific focus on business apps [43]. In the following, we provide the rationale for each of our criteria following the above proposed categorisation. Besides referencing sources already discussed, we provide additional evidence where appropriate¹.

4.2 Infrastructure Perspective

(I1) License: The license under which a framework is published is essential for the type of product to develop. It needs to be assessed whether a developer is free to create commercial apps, for example when using open source software [11, 13, 47]. In addition, the pricing model needs to be considered. A framework could be freely distributed, or require one-time or regular license payments with regard to the number of developers, projects, or as a flat fee [32, 61].

(I2) Supported Target Platforms: The number and importance of supported mobile platforms within a device class is a major concern for choosing a cross-platform approach [11, 47]. Furthermore, support varies regarding different versions of each mobile operating system. The most recent version provides the newest features and its support is important to reach early adopters of a new technology [5]. However, the majority of users will use an old version of the system due to hardware limitations or slow update behaviour by users or vendors [17]. Finally, it needs to be considered whether multiple device classes have to be bridged, for example a combination of smart TV and tablet application.

(I3) Supported Development Platforms: Flexibility regarding supported development platforms is beneficial for heterogeneous teams in which developers are accustomed to specific hardware and software such as an development environment [47] (see also criterion D1). Moreover, the role within a team such as UI or UX design may require the approach to support multiple platforms.

(I4) Distribution Channels: With proprietary platform- or vendor-specific app stores typical for publication, the number of users who can be reached is critical. It needs to be weighted against fixed and variable costs of app store accounts and app publishing. The ease of the publication process itself also

¹ However, we do *not* cite [28] for each single criterion originating from this work.

needs to be considered, regarding e.g. the average duration for initial app placement and update distribution as well as the strictness and detailedness of the review process [68]. Cross-platform frameworks vary by the degree of compatibility with app store restrictions and submission guidelines [16, 61]. Further app store integrations include app rating to reach a better app store ranking as well as automatic update notifications within the app for rolling out updates fast [32].

(I5) Monetisation: From a business perspective, the possibility and the complexity of selling the app itself and subsequent in-app purchases need to be considered as well as the availability of advertisement [16]. These features need to be traded off against direct costs and commissions to the app store operator. Again, cross-platform development frameworks can support this aspect, for example by providing interfaces to payment providers or advertising networks.

(I6) Global App Distribution: Typically, a global distribution of apps is desired – unless specific reasons for restrictions exist. Approaches can offer built-in support for internalisation and localisation to create and distribute app versions targeted (and potentially restricted) to specific geographic regions. In addition, translation capabilities allow for easy delivery of multi-language content and provide format conversions for dates, currencies and location particularities [61].

(I7) Long-term Feasibility: Choosing an approach might be an important strategic decision considering the significant initial investment for training or hiring developers as well as the risk of technology lock-in, particularly for smaller companies. The maturity and stability of a framework can be evaluated concerning the historical and expected backwards incompatible changes of major releases. Other indicators are short update cycles, regular bug-fixes, and security updates. In an active community, developers exchange knowledge to solve issues. Ideally, several commercial supporters back the project with financial resources and steady contribution. Costs for professional support inquiries need also be considered, potentially increasing the attractiveness of a promising open-source project while safeguarding efficient solutions to development issues [32, 61].

4.3 Development Perspective

(D1) Development Environment: The maturity and features of an integrated development environment (IDE) heavily influence development productivity and speed. Tool support includes functionalities of the IDE such as auto-completion, debugging tools, and an emulator to enable rapid app development cycles [11, 16, 32, 47, 61]. In addition to the IDE typically associated with the cross-platform approach, the freedom to use accustomed workflows, e.g. choose a preferred IDE, lowers the initial set-up effort for additional dependencies such as runtime environments or software development kits (SDK) [61].

(D2) Preparation Time: The learning curve, i.e. the subjective progress of a developer while exploring the capabilities and best practices of the approach, should foster rapid initial progress. To lower the entry barrier, the number and type of required technology stack and programming languages need to be considered [11, 47, 61, 68], e.g. using known paradigms to further reduce the initial

learning efforts [11]. With unique benefits and characteristics, the quality of API documentation is also of major importance. “Getting started” guides, tutorials, and code examples initially clarify the framework’s features and structure, whereas a corpus of best practices, user-comments, and technical specifications support in solving issues over the course of development [16,61].

(D3) Scalability: Scalability refers to the modularisation capabilities of the framework and generated apps in large-scale development projects. Partitioning code in subcomponents and architectural design decisions such as the well-known Model-View-Controller pattern has implications on the app structure. Thus, the number of developers can be increased while extending the app’s functionality [32,47]. However, a modular framework itself can guide and support this division of labour. With specified interfaces and interactions between the components, developers can specialize themselves on few relevant components.

(D4) Development Process Fit: Departing from the traditional approach of implementing software from a fixed and comprehensive specification, a variety of methods with agile characteristic exist today. For such projects, the cross-platform approach can be evaluated regarding the effort to create the *minimum viable product*, e.g. the amount of boilerplate code and initial configuration, as well as the effort to subsequently modify its scope. This criterion also relates to the organisational aspect of scalability in terms of developer specialisation. In contrast to full-stack developers in small projects, modularizing development using roles can be supported through tailored views or specialized tools [64].

(D5) UI Design Approach: UI development is a major concern for cross-platform approaches. Graphical user interfaces are highly platform-specific and often just covered by a default appearance defined by the framework [29]. In addition, a separate WYSIWYG editor to develop appealing interfaces for multiple devices can be beneficial and increases the speed of development compared to constantly deploying the full app to a device or an emulator.

(D6) Testing Support: App logic and user interfaces need to be tested with established concepts such as system and unit tests [32,61]. To test context-sensitive mobile scenarios more authentically, external influences (such as *bad connectivity*) may be simulated [42]. Furthermore, possibilities of monitoring the app at runtime improve the testability, e.g. providing a developer console, meaningful error reporting, and logging functionalities for app-specific and system events. Tool support may also include remote debugging on a connected device rather than emulator environments, test coverage visualisation and metrics [32].

(D7) Deployment Support: Build toolchain support immensely simplifies the deployment process, i.e. generating individual packages for all targeted platforms. Approaches vary from requiring all native SDKs to external build services and cloud-based techniques [32,61]. Sophisticated projects additionally use *continuous integration* platforms to automate testing. Frameworks can be explicitly designed to integrate with such toolchains. Regarding production, the framework might also offer optimised build options (e.g. *minified* code) and app store integrations to automatically publish updates [32].

(D8) Maintainability: In contrast to (I7), maintainability deals with the evolution of a code base over time [61] and difficult to quantify. Lines of code (LOC) for a specified reference app may be used for comparison with the assumption that less LOC are easier to support regarding readability of source code, amount of training and familiarisation, etc. This concept is similar to programming languages themselves, where so-called *gearing factors* try to compare the amount of code per unit of functionality [23]. Advanced maintainability metrics are hard to apply due to the heterogeneity and varying complexity of frameworks, especially in case of apps composed of different programming languages. Furthermore, the reusability of source code across development projects can be evaluated, for instance concerning the portability to other software projects [61].

(D9) Extensibility: Special requirements may introduce the need for features that go beyond the core of the framework. These might not be put into practice with high priority. Therefore, the possibility to extend the framework with custom components and third-party libraries should be evaluated. Examples for such extensions include additional UI elements, functionalities to access device features, and solutions to common challenges such as data transfer [32, 47].

(D10) Integrating Native Code: For some applications, running native code within the application is a requirement. This seemingly invalidates the idea of cross-platform development; nonetheless, it *can* be beneficial: Previously existing code can be reused, e.g. when migrating apps to a cross-platform development approach and replacing platform-specific code over time. Also, native platform APIs might enable access to platform functionalities and device features currently not available on the framework’s level of abstraction [47, 61].

(D11) Speed of Development: Rapid development is influenced by the amount of boilerplate code necessary for functional app skeletons (cf. [30]) and the availability of typical app functions such as user authentication. Assuming salaries to be independent of programming language proficiency, development speed directly influences the variable costs and ultimately the return-on-investment.

4.4 App Perspective

(A1) Access to Device-specific Hardware: For cross-platform approaches, the coverage of platform- and device-specific hardware is of supreme importance [11, 13, 16, 32, 61]. Especially regarding the capabilities of novel mobile devices, a plethora of device hardware is present today. This includes sensors such as camera, microphone, GPS, accelerometer, gyroscope, magnetometer, temperature sensor, and heart rate monitor. In addition, cyberphysical systems enable bidirectional interaction that can modify the environment through actuators. The set of individual features is evaluated according to the framework’s documentation.

(A2) Access to Platform-specific Functionality: Regarding the software side of the various mobile platforms, functionalities include a persistence layer such as the file system and access to a database on the device, contact lists, information on the network connection, and battery status [16, 32]. In addition,

in-app browser support may be desirable for fetching additional content from the Internet without leaving the app [32]. Advanced features like monitoring or push notifications can be realised using background services [61].

(A3) Support for Connected Devices: Current wearable devices and also sensor/actuator networks of cyberphysical systems often require to be coupled to a respective master device (e.g. a smartphone). This trend of “device extensions” needs to be evaluated regarding viable device combinations. More specific this includes the level of access to coupled device data and sensors as well as additional user interfaces. This may be trivial if the platform provides a layer of abstraction that exposes the coupled device similar to other device components. Yet, in many cases the cross-platform approach needs to take care of this additional complexity.

(A4) Input Device Heterogeneity: The input device criterion evaluates the support of the approach with regard to the variety of input devices that can be used to interact with the app. This includes traditional devices such as keyboard and mouse as well as (multi-) touch screens, voice recognition, remote controls, hardware buttons and more. Each of these devices can process many interaction mechanisms, for example multi-touch screens reacting to gestures such as different types of taps, swipes, pinches, pressure, orientation changes etc., all of which the cross-platform tool needs to make available to the app developer.

(A5) Output Device Heterogeneity: Mobile devices provide a huge variety of different output devices differing in device size, resolution, format (e.g. *round* smartwatches), colour palette, frame rate (e.g. E-Ink screens), and opacity (e.g. augmented reality projections). This poses challenges as adaptability is already a major challenge for traditional devices [1]. In addition, the app has to adapt to device class-specific context changes, thereby realizing well-understood design ideals [58] such as day/night-mode appearances for in-vehicle apps.

(A6) Application Life Cycle: This criterion refers to how far a framework supports the life-cycle inherent to an app. Platforms may differ in starting, pausing, continuing, and exiting an app [61]. Additional differences arise from the life cycle of individual views and view elements.

(A7) Business Integration: To integrate with the overall business, support for data exchange protocols, serialisation, and multiple data formats are often required [13]. Apps may communicate with existing Web service back-ends for data storage and processing, or initiate inter-app communication. E.g., business processes often require collaboration of different user roles. Business integration also refers to customizability, e.g. being adaptable to a *corporate identity* [61].

(A8) Security: Frameworks can support the development of secure apps on several levels. First, mobile platforms are usually restrictive regarding access permissions. Requesting permissions on demand increases not only the perceived security of an app. Second, data loss can be avoided by using data encryption mechanisms on the device as well as secure data transfer protocols against eavesdropping [13, 32]. Third, the framework may provide user input validation and prevent cross-site forgery and code injection [32].

(A9) Fallback Handling: Considering the device and platform heterogeneity of (A1)–(A5), intelligent fallback mechanisms aid in case individual features are unsupported or restricted. As a naïve approach, the user may be redirected to a Web page. Sophisticated actions include *graceful degradation* techniques with simpler representations [22], or alternative functions to fulfil the user’s task.

4.5 Usage Perspective

(U1) Look and Feel: This criterion considers whether available UI elements have a native look & feel or rather behave like a Web site [61]. The set of elements can be evaluated according to the human interface guidelines of the respective platform. Particularly, rich user interfaces with 2D/3D animation and multimedia features are challenging for cross-platform tools [13]. In addition, it should be considered to which degree a framework supports the platform-specific usage philosophy, e.g. the position of navigation bars, scrolling, and gestures [61].

(U2) Performance: Application speed, stability, and responsiveness of the app on user interaction are essential performance aspects. Apart from the subjective user experience, the speed at start-up, after interruptions and for shut-down can be measured [16]. Moreover, resource usage can be assessed, e.g. CPU, RAM and battery utilisation at runtime, or download size [10, 11, 13, 61].

(U3) Usage Patterns: Apps are frequently used for a short amount of time and are likely to be interrupted. Users want an “instant on” experience and continue where they left the app. To match usage patterns, apps have to integrate into personal workflows for information processing such as sharing with other apps or saving to persistent storage, and community interactions such as messaging, e-mail, and social media. For some use cases, support for synchronisation of app data across multiple devices of the user for seamless context switching is beneficial. In addition, notification centres of the platform are gaining importance for app interaction.

(U4) User Management: Cross-platform frameworks may support different types of user handling, reaching from purely local apps to user accounts across multiple devices and role-based authentication. Authentication may therefore be performed in-app or server-based, and potentially connected to session management. In addition, mobile devices may provide various login mechanisms, including traditional passwords, gestures, and biometric information such as fingerprints, voice recognition, or other characteristics [40].

5 Evaluation

Due to their recent emergence, cross-platform approaches barely exist for novel mobile device classes (cf. Sect. 2). Therefore, the following evaluation compares PhoneGap to Web apps and native applications with regard to traditional smartphone mobile devices. PhoneGap was chosen due to its perennial popularity as leading cross-platform development tool [12]. The evaluation is by no means a comprehensive survey of the cross-platform framework itself (as provided

by [16,32]) but should serve as exemplary comparison in order to discuss our approach of weighted criteria evaluation. Thereby, this evaluation particularly serves as a benchmark for our evaluation framework.

5.1 Weight Profiles

To cater for differences across heterogeneous and evolving mobile device classes, our approach to cross-platform tool evaluation applies a weighting mechanism. Each of the 31 criteria receives between 1 and 7 points with a total of 100 points assigned (not necessarily distributed equally across the categories), constituting the so-called *weight profile*. Each criterion is evaluated on a scale from 0 (criterion unsatisfied) to 5 (optimally fulfilled). The weighting points directly translate to percental values used in calculating the *weighted score*. A weight profile reflects the requirements of a specific device class regarding cross-platform development. It can be individually adapted to the future evolution of the mobile ecosystems, as well as changed to reflect particular needs, e.g. regarding the background of developers. The proposed weight profiles for the device classes presented in Sect. 3 are depicted in Table 2 along with an exemplary evaluation.

In the following, we focus on the *smartphone* device class, which can be backed with empirical and theoretical work. Studies have shown that cross-platform approaches are often developer-oriented [12,52]. From an infrastructure perspective, this means that free and open approaches are considered particularly important. Long-term feasibility benefits from a stabilized smartphone ecosystem with Android and iOS as main players [67]. Distribution channels are mostly limited to platform-specific app stores with a broad set of features.

App developers want to use existing standards and previous knowledge for fast-paced development [12,52]. In contrast, the current practice of smartphone apps apparently does not cover large development teams. As a result, organisational aspects such as scalability, maintainability, and development process integration are not requested by practitioners [52]. UI design seems to be an ongoing challenge for cross-platform frameworks and may even become more important for “standing out from the mass of apps” [1].

On the application side, access to a broad range of device functionalities is requested, while support for smartphone screens as both input and output device has matured [12]. Apps are still rather developed for social and communication purposes [37], thus business integration and security issues are not prioritized.

With mobile usage soon surpassing desktop usage [37], performance and native look and feel remain important topics for smartphone development. Finally, user management and usage patterns play an inferior role on smartphones as these are mainly designed for single-person usage.

5.2 Web Apps

Web apps are mobile-optimized Web sites built with HTML5 and JavaScript (JS), and executed within the smartphone’s browser. They rely on

Table 2. Comparison of approaches and device class weight profiles

Smartphone comparison					Category weights				
Criterion	Weight (%)	Web apps	PhoneGap	Native apps	Tablets	Entertainment	Wearables	Vehicle	Smart home
I1: License	6	5	5	5	5	6	5	3	5
I2: Target platforms	7	5	5	1	5	6	7	4	7
I3: Development platforms	2	5	5	2	2	2	1	1	1
I4: Distribution channels	2	5	3	4	2	3	4	3	3
I5: Monetisation	2	0	3	5	2	1	1	2	2
I6: Global distribution	2	1	3	5	2	2	2	0	1
I7: Long-term feasibility	5	5	5	4	5	3	3	6	5
D1: Dev. environment	7	4	5	5	7	7	5	5	6
D2: Ramp-up time	7	5	4	3	7	7	5	1	5
D3: Scalability	2	3	3	3	2	3	2	3	2
D4: Development process fit	2	3	4	2	2	3	1	4	2
D5: UI design	4	3	3	4	4	5	5	6	3
D6: Test support	3	3	4	5	3	3	4	7	3
D7: Deployment support	3	5	5	3	3	3	4	5	2
D8: Maintainability	2	2	4	2	2	2	1	5	2
D9: Framework extensibility	2	5	5	0	2	2	2	1	2
D10: Native extensibility	2	0	3	5	2	2	1	0	0
D11: Speed of development	4	2	3	0	4	3	3	2	4
A1: Hardware access	5	2	4	5	3	1	6	4	7
A2: Platform functionality	5	2	4	5	5	3	2	3	3
A3: Connected devices	3	0	0	5	2	1	7	4	7
A4: Input heterogeneity	1	4	4	5	3	3	2	2	2
A5: Output heterogeneity	1	4	4	5	1	1	6	3	4
A6: App life cycle	3	0	4	5	3	3	3	3	2
A7: Business integration	2	3	3	5	3	3	1	2	1
A8: Security	3	0	0	1	4	1	3	7	5
A9: Fallback Handling	2	2	4	0	1	4	3	2	1
U1: Look and feel	4	1	3	5	4	2	4	5	3
U2: Performance	4	3	2	5	4	6	3	3	2
U3: Usage patterns	2	0	1	2	4	4	4	3	4
U4: User management	1	0	0	0	2	5	0	1	4
Weighted score		2.99	3.66	3.56					

open standards and are highly cross-platform compatible while using Web development tools (I1–I4). While profiting from an immense community of developers, “app-like” behaviour needs to be implemented manually and distribution cannot be controlled (I5–I6).

Only Web development skills are required; many tutorials and profound tool support is available (D1–D2). The universality of the Web at the same time limits the application to apps, e.g. requiring boilerplate code or providing no guidance on the structure of source code and development (D3–D5). Testability is problematic: desktop browsers emulate the respective mobile counterpart inconsistently and mobile in-browser debugging is hardly supported (D6). Various libraries simplify development, yet native code is unsupported (D9–D10). As a result, Web apps are rather easy to create and modify using established toolchains. However, all aspects regarding app life cycle, integration, security, and fallback have to be built manually without platform-specific abstraction (D7–D8, D11, A6–A9).

Accessing device components is possible only via HTML5 APIs such as Media Capture Stream, which are scarcely supported by mobile browsers (A1–A2) [44]. As the execution happens in the browser, keyboard and gesture support are well established through JS events but limited to Web page behaviour and browser controls (A3, U1). Furthermore, CSS can be used to customise the design and target different output devices, with the exception of connected devices (A4–A5, A7). Finally, usage patterns and user management are completely up to the developer, whereas the overall performance depends on the smartphone browser and is likely to be optimized by the platform provider (U2–U4).

5.3 PhoneGap

PhoneGap was initially developed around 2009 and is still the top-used cross-platform development tool [12, 14]. Freely available under the permissive Apache License, PhoneGap targets all major smartphone platforms in various versions (I1). Technically, apps are developed using HTML5/JS/CSS and executed within a *Web view wrapper* component without browser controls. This allows installing the apps and providing API access to native functionality. Thus, the framework does not adhere to specific platform guidelines but provides a general mobile appearance that can be distributed through any app store but without advanced features such as in-app purchases (I4–I6). Its long existence and stable API has created a large community that in turn supports the long-term perspective (I7).

Similar to Web apps, developers can freely choose their preferred Web development environment and profit from previous knowledge. The framework’s structure requires little knowledge and it is well documented (I3, D1–D2). PhoneGap generates a running app skeleton and file structure but does not impose further implications on the development process (D3–D5). All app functionality needs to be implemented manually (A8–A9, U3–U4). Outstanding features are the *cloud deployment* that requires no locally installed SDKs as well as the *remote debugging interface* that connects to real devices (D6–D7). Maintainability is

enabled through the extensive and stable API abstracting from platform differences and increasing the speed of development (D8, D11, A1–A2). In addition, numerous plug-ins exist, extending the functionality and covering many of the aforementioned drawbacks, also allowing the execution of native code (D9–D10).

Regarding native behaviour and appearance, the *Event API* provides access to life cycle events and platform settings can be retrieved via the *Device API* to target specific platforms (A6–A7, A9, U1). Support for input and output is similar to Web apps and likewise restricted to the main device (A3–A5). The app performance depends again on the smartphone’s browser capabilities with additional framework overhead (U2) [4, 50].

5.4 Native Apps

Any cross-platform development approach can be benchmarked against native app development. While it naturally is closest to a platform’s capabilities, developing natively not necessarily is the most efficient or elegant option.

Platform SDKs are freely available and fully integrated into the respective app stores. The latter provide a broad set of features for distribution and monetisation (I1, I4–I6). Whereas development might be possible with several technologies, the target platforms are limited to one (I2–I3).

iOS and Android as prevailing platforms and can be treated as reliable on long-term [67]. Platforms typically require specific programming language knowledge, although extensive documentation and community support are available (D2). Moreover, a full ecosystem with tool support for all phases of development is usually provided, with varying degrees of alternatives (D1, D5–D7).

The flexibility of implementation comes at the cost of few guidelines on structuring and subdividing development work (D3–D4). The platforms usually do not provide support for recurring programming tasks (A8–A9, U3–U4). Obviously, the speed of developing multiple native apps is unmatched low (D11).

Native apps can access all possible features of a given platform (A1–A7). Ultimately, a fully native appearance and behaviour as well as performance without runtime overhead can only be reached with native apps (A1–A2).

6 Discussion

The framework presented in the prior sections should provide a step towards a sound theory of cross-platform app development. However, it is by no means static. In fact, we hope it can be the foundation for application and extension by others. Thereby, the framework can stay at eye level with further developments in the field. Specifically, depending on the emergence of novel device classes and the possible proliferation of further kinds of devices, revisions can be applied.

In the following we reflect on our work, starting with a *synthesis of findings*. Our criteria have proven to be useful and applicable in the exemplary use. Categorisation into four perspectives worked well, although it remains to be seen

whether an even finer scheme might be advisable to cater for future developments. While the weighting profiles will need further tweaking (see also below), they lead to producible results. In particular, the smartphone profile has proven to be feasible. As could be expected and is widely affirmed by related literature (cf. Sects. 2 and 3), PhoneGap as the leading approach is better suited for cross-platform development that targets smartphones than pure native or Web apps. It should be noted that native development not simply satisfies all criteria *but* cross-platform capabilities; working natively can have its own *overhead*.

The additional weight profiles for now have to be seen as proposals. They should nonetheless be reasonable starting points. In particular, they are well-suited to address idiosyncrasies of specific device classes, e.g. to put weight on security for apps in cars or smart homes.

The tablet profile is rather similar to smartphones, particularly from the infrastructure and the development perspective. Multi-user scenarios and business integration need more attention, and additional means of input play a role. Quite differently, the entertainment profile has less business implications and is less focused on security, sensors and platform-features. Performance requirements might add complexity and support for multiple users is a prerequisite.

For the wearable profile, yet other specialities need to be taken into account. Deploying to and testing on devices is quite hard. User interfaces differ much from platform to platform. Apps typically are very small and must perform with low resource utilisation. At the same time, usage scenarios are simpler and due to high-fluctuation of devices a long-term focus needs to be less emphasized.

Security is of foremost concern in the vehicle profile. Due to the field's fuzziness, it shares similarities with the wearable profile but has more focus on professional software development. Most blurry is the smart home profile, which needs to address the heterogeneity of possible devices along with security concerns.

A number of open questions can be raised. While we deem the evaluation framework to be readily usable, particularly due to its solid literature foundation, the weighting remains open for revisions. Future research will scrutinize whether the device classes have been chosen wisely. There is no easy answer to this since new kinds of devices might be designed with a focus on app-enablement – or not. For example, Tesla announced an own SDK but current work obviously has taken another direction due to security concerns [36]. Moreover, it is hard to predict market development. For example, Android Wear [3] *might* unify development for Wearables or at least consolidate different streams.

It remains to be seen whether the success of Web technology (including frameworks such as PhoneGap) will be repeated for new device classes. On the one hand, devices with hardware that is not powerful enough to run a WebKit-Engine such as some watches might require different approaches. Other devices, such as arguably fitness trackers, do not even pose a platform that would be comparable to Android or iOS. On the other hand, Web technology might be the bridging element for heterogeneity. It is still very hard to image the proper abstraction for devices that fall under the umbrella of smart home technology.

Furthermore, it needs to be questioned whether for all device classes full ecosystems as for smartphones will be established. A Cloud-based middleware, mirroring, or other “remote” approaches could solve issues such as low performance, hardware heterogeneity and security without even relying on devices directly. Moreover, device classes might converge. Modern fitness trackers have smartwatch functionality; a smartwatch was recently hacked to run applications only imaginable on smartphones before [35]. So called *instant apps* can be run without installation [24] and might also contribute to future changes.

Due to the breadth of our work and also due to the novelty of some of the tackled topics, this paper is bound to limitations. While we built upon the literature both for the derivation of criteria and for their exemplary usage, we have not evaluated our work empirically. This is particularly an issue for the weight profiles, which need to be assessed based on the input from practitioners. Moreover, we have made assumptions about the future, most notably considering device classes. It seems unlikely but it might turn out that e.g. app-enabled cars will not gain importance. Even if they do, it is not given that cars (or other device classes) will allow for reasonable cross-platform app development support. Looking towards the future is part of our work but a boundary at the same time.

The limitations do not impede the value of our work, though. In fact, in combination with the above discussed open questions they provide the foundation for our future work. Writing this paper has been *more* than setting out to refresh the view on the topic of mobile computing. It has brought up a host of new ideas for us, particularly revolving around the differences in device classes. We will strive to provide a unified understanding while honouring the particular strengths and possibilities offered by devices. A major source of our future work will be the above mentioned limitations. As a next step, we will work on a broader evaluation of current approaches based on our criteria. Moreover, we will assess possibilities how to get practitioners’ feedback on the framework, ideally leading to an empiric validation of our work. This will include a revision of the weights and more concrete advice on approach choice. In particular, we would like to provide recommendations in form of case study-like scenarios for future applications. Finally, we will also seek to make further theory contributions, especially concerning an abstraction from device classes.

7 Conclusion

In this paper, we have presented work on an extended cross-platform app development evaluation framework. It extends existing papers and revised the criteria formerly proposed. In particular, it takes into account differences in the increasing number of device classes and provides a weighted evaluation. We have not only comprehensively introduced our framework but given an exemplary evaluation. The findings suggest that the framework is well-suited. Nonetheless, much work remains due to the novelty and breadth of the field.

References

1. Amatya, S., Kurti, A.: Cross-platform mobile development: challenges and opportunities. In: Trajkovik, V., Anastas, M. (eds.) *ICT Innovations 2013. AISC*, vol. 231, pp. 219–229. Springer, Heidelberg (2014). doi:[10.1007/978-3-319-01466-1_21](https://doi.org/10.1007/978-3-319-01466-1_21)
2. Android TV. <https://www.android.com/tv/>
3. Android Wear 2.0 developer preview. <https://developer.android.com/wear/preview/index.html>
4. Apache Cordova documentation (2016). <https://cordova.apache.org/docs/en/>
5. Beal, G.M., Bohlen, J.M.: *The Diffusion Process*. Agricultural Experiment Station. Iowa State College, Ames (1957)
6. Bouhnick, G.: A list of all operating systems running on smart-watches [wearables] (2015). <http://www.mobilespoon.net/2015/03/a-list-of-all-operating-systems-running.html>
7. Rahul Raj, C.P., Tolety, S.B.: 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 (2012)
8. Carter, J.: Which is the best internet of things platform? (2015). <http://www.techradar.com/news/-1302416>
9. Chmielewski, J.: Towards an architecture for future internet applications. In: Galis, A., Gavras, A. (eds.) *FIA 2013. LNCS*, vol. 7858, pp. 214–219. Springer, Heidelberg (2013). doi:[10.1007/978-3-642-38082-2_18](https://doi.org/10.1007/978-3-642-38082-2_18)
10. Ciman, M., Gaggi, O.: Measuring energy consumption of cross-platform frameworks for mobile applications. In: Monfort, V., Krempels, K.-H. (eds.) *WEBIST 2014. LNBIP*, vol. 226, pp. 331–346. Springer, Heidelberg (2015). doi:[10.1007/978-3-319-27030-2_21](https://doi.org/10.1007/978-3-319-27030-2_21)
11. Ciman, M., Gaggi, O., Gonzo, N.: Cross-platform mobile development: a study on apps with animations. In: *Proceedings of the ACM Symposium on Applied Computing* (2014)
12. Cross-platform tools 2015 (2015). <http://www.visionmobile.com/product/cross-platform-tools-2015/>
13. Dalmasso, I., Datta, S.K., Bonnet, C., Nikaein, N.: Survey, comparison and evaluation of cross platform mobile application development tools. In: *Proceedings of the 9th IWCMC* (2013)
14. Davis, L.: Phonegap: people’s choice winner at web 2.0 expo launch pad (2009). http://readwrite.com/2009/04/02/phone_gap
15. Deindl, M., Roscher, M., Birkmeier, M.: An architecture vision for an open service cloud for the smart car. In: Filho, W.L., Kotter, R. (eds.) *Mobility in Europe, Green Energy and Technology*, vol. 203, pp. 281–295. Springer, Heidelberg (2015)
16. Dhillon, S., Mahmoud, Q.H.: An evaluation framework for cross-platform mobile application development tools. *Softw. Prac. Exp.* **45**(10), 1331–1357 (2015)
17. Dobie, A.: Why you’ll never have the latest version of android (2012). <http://www.androidcentral.com/why-you-ll-never-have-latest-version-android>
18. Dorr, T.: Tesla Model S JSON API (2016). <http://docs.timdorr.apiary.io>
19. Doud, A.: How important is cross-platform wearable support? (2015). <http://pocketnow.com/2015/05/10/cross-platform-wearable-support>
20. Durach, S., Higgen, U., Huebler, M.: Smart automotive apps: an approach to context-driven applications. In: *SAE-China, FISITA* (ed.) *Proceedings of the FISITA 2012 World Automotive Congress. LNEE 2012*, vol. 200, pp. 187–195. Springer, Heidelberg (2013). doi:[10.1007/978-3-642-33838-0_17](https://doi.org/10.1007/978-3-642-33838-0_17)

21. El-Kassas, W.S., Abdullah, B.A., Yousef, A.H., Wahba, A.: ICPMD: integrated cross-platform mobile development solution. In: Proceedings of the 9th ICCES (2014)
22. Ernsting, J., Rieger, C., Wrede, F., Majchrzak, T.A.: Refining a reference architecture for model-driven business apps. In: Proceedings of the 12th WEBIST, pp. 307–316. SciTePress (2016)
23. Function point languages table: Version 5.0 (2009). <http://www.qsm.com/resources/function-point-languages-table>
24. Ganapathy, S.: Introducing android instant apps. <http://android-developers.blogspot.no/2016/05/android-instant-apps-evolving-apps.html>
25. Google Inc.: Android wear 2.0 developer preview. <https://developer.android.com/wear/preview/index.html>
26. HbbTV overview (2016). <https://www.hbbtv.org/overview/>
27. Heitkötter, H., Hanschke, S., Majchrzak, T.A.: Comparing cross-platform development approaches for mobile applications. In: Proceedings 8th WEBIST, pp. 299–311. SciTePress (2012)
28. Heitkötter, H., Hanschke, S., Majchrzak, T.A.: Evaluating cross-platform development approaches for mobile applications. In: Cordeiro, J., Krempels, K.-H. (eds.) Web Information Systems and Technologies. LNBIP, vol. 140, pp. 120–138. Springer, Heidelberg (2013). doi:[10.1007/978-3-642-36608-6_8](https://doi.org/10.1007/978-3-642-36608-6_8)
29. Heitkötter, H., Majchrzak, T.A., Kuchen, H.: Cross-platform model-driven development of mobile applications with MD². In: Proceedings of the SAC 2013, pp. 526–533. ACM (2013)
30. Heitkötter, H., Majchrzak, T.A., Ruland, B., Weber, T.: Comparison of mobile web frameworks. In: Krempels, K.-H., Stocker, A. (eds.) Web Information Systems and Technologies. LNBIP, vol. 189, pp. 119–137. Springer, Heidelberg (2014)
31. Heitkötter, H., Kuchen, H., Majchrzak, T.A.: Extending a model-driven cross-platform development approach for business apps. Sci. Comput. Program. **97**(Part 1), 31–36 (2015)
32. Hudli, A., Hudli, S., Hudli, R.: An evaluation framework for selection of mobile app development platform. In: Proceedings of the 3rd MobileDeLi (2015)
33. Jie, G., Bo, C., Shuai, Z., Junliang, C.: Cross-platform android/ios-based smart switch control middleware in a digital home. Mobile Inform. Sys. (2015). <http://www.hindawi.com/journals/misy/2015/627859/>
34. Kim, H., Ahn, M., Hong, S., Lee, S.: Wearable device control platform technology for network application development. Mobile Inform. Syst. (2016). <http://www.hindawi.com/journals/misy/2016/3038515/>
35. Krawczyk, K.: Hacker installs windows 95 and doom on a samsung gear live smartwatch. <http://www.digitaltrends.com/computing/hacker-installs-windows-95-and-doom-on-a-samsung-gear-live-smartwatch/>
36. Lambert, F.: Tesla is moving away from an SDK. <http://9to5mac.com/2016/01/28/tesla-sdk-iphone-apps-mirror/>
37. Lella, A., Lipsman, A., Martin, B.: The 2015 U.S. mobile app report. <https://www.comscore.com/ger/Insights/Presentations-and-Whitepapers/2015/The-2015-US-Mobile-App-Report>
38. LG Electronics: WebOS for LG smart TVs (2016). <http://www.lg.com/uk/smarttv/webos>
39. Liu, X., Vega, K., Maes, P., Paradiso, J.A.: Wearability factors for skin interfaces. In: Proceedings of the 7th Augmented Human International Conference, pp. 21:1–21:8. ACM (2016)

40. Luca, A.D., Lindqvist, J.: Is secure and usable smartphone authentication asking too much? *Computer* **48**(5), 64–68 (2015)
41. Majchrzak, T.A., Ernsting, J.: Reengineering an approach to model-driven development of business apps. In: Wrycza, S. (ed.) SIGSAND/PLAIS 2015. LNBIP, vol. 232, pp. 15–31. Springer, Heidelberg (2015). doi:[10.1007/978-3-319-24366-5_2](https://doi.org/10.1007/978-3-319-24366-5_2)
42. Majchrzak, T.A., Schulte, M.: Context-dependent testing of applications for mobile devices. *Open J. Web Technol. (OJWT)* **2**(1), 27–39 (2015)
43. Majchrzak, T.A., Wolf, S., Abbassi, P.: Comparing the capabilities of mobile platforms for business app development. In: Wrycza, S. (ed.) SIGSAND/PLAIS 2015. LNBIP, vol. 232, pp. 70–88. Springer, Heidelberg (2015). doi:[10.1007/978-3-319-24366-5_6](https://doi.org/10.1007/978-3-319-24366-5_6)
44. MobileHTML5: Mobile html5 compatibility (2015). <http://mobilehtml5.org/>
45. Noreikis, M., Butkus, P., Nurminen, J.K.: In-vehicle application for multimodal route planning and analysis. In: Proceedings of the IEEE 3rd CloudNet (2014)
46. Ohrt, J., Turau, V.: Cross-platform development tools for smartphone applications. *Computer* **45**(9), 72–79 (2012)
47. Palmieri, M., Singh, I., Cicchetti, A.: Comparison of cross-platform mobile development tools. In: Proceedings of the 16th ICIN, pp. 179–186. IEEE (2012)
48. Perakakis, E., Ghinea, G.: HTML5 technologies for effective cross-platform interactive/smart TV advertising. *IEEE Trans. HMS* **45**(4), 534–539 (2015)
49. Perakakis, E., Ghinea, G.: A proposed model for cross-platform web 3D applications on smart TV systems. In: Proceedings of the 20th Web3D (2015)
50. Phonegap documentation (2015). <http://docs.phonegap.com>
51. Quaresma, M., Gonçalves, R.: Usability analysis of smartphone applications for drivers. In: Marcus, A. (ed.) DUXU 2014. LNCS, vol. 8517, pp. 352–362. Springer, Heidelberg (2014). doi:[10.1007/978-3-319-07668-3_34](https://doi.org/10.1007/978-3-319-07668-3_34)
52. Research2guidance: cross-platform tool benchmarking (2014). <http://research2guidance.com/product/cross-platform-tool-benchmarking-2014/>
53. Revest, F.: Asteroidos (2016). <http://asteroidos.org/>
54. Rodriguez Garzon, S., Poguntke, M.: The personal adaptive in-car HMI: integration of external applications for personalized use. In: Ardissono, L., Kuflik, T. (eds.) UMAP 2011. LNCS, vol. 7138, pp. 35–46. Springer, Heidelberg (2012). doi:[10.1007/978-3-642-28509-7_5](https://doi.org/10.1007/978-3-642-28509-7_5)
55. Ryu, D., Krompiec, P.K., Lee, E., Park, K.: A serious game design for english education on smart TV platform. In: Proceedings of the ISCE (2014)
56. Samsung Electronics Co. Ltd.: Let's toast - samsung smart TV apps developer forum. <https://www.samsungdforum.com/Features/TOAST>
57. Sansour, R.N., Kafri, N., Sabha, M.N.: A survey on mobile multimedia application development frameworks. In: Proceedings of the ICMCS (2014)
58. Schilit, B., Adams, N., Want, R.: Context-aware computing applications. In: Proceedings of the 1994 1st WMCSA, pp. 85–90. IEEE CS (1994)
59. Schuermans, S., Vakulenko, M.: Apps for connected cars? Your mileage may vary (2014). <http://www.visionmobile.com/product/apps-for-cars-mileage-may-vary/>
60. Sohn, H.J., Lee, M.G., Seong, B.M., Kim, J.B.: Quality evaluation criteria based on open source mobile HTML5 UI framework for development of cross-platform. *IJSEIA* **9**(6), 1–12 (2015)
61. Sommer, A., Krusche, S.: Evaluation of cross-platform frameworks for mobile applications. LNI P-215 (2013)
62. Statista. <http://www.statista.com/>
63. Tizen (2016). <https://www.tizen.org/>

64. Wasserman, A.I.: Software engineering issues for mobile application development. In: Roman, G.C., Sullivan, K. (eds.) Proceedings of the FoSER 2010, p. 397 (2010)
65. Willocx, M., Vossaert, J., Naessens, V.: A quantitative assessment of performance in mobile app development tools. In: Proceedings of the 3rd International Conference on Mobile Services (2015)
66. Wolf, F.: Will vehicles go the mobile way? Merits and challenges arising by car-apps. In: Proceedings of the 10th ICINCO, vol. 2 (2013)
67. Woods, V., van der Meulen, R.: Gartner says worldwide smartphone sales grew 9.7 percent in fourth quarter of 2015 (2016). <http://www.gartner.com/newsroom/id/3215217>
68. Xanthopoulos, S., Xinogalos, S.: A comparative analysis of cross-platform development approaches for mobile applications. In: Proceedings of the 6th BCI, pp. 213–220. ACM (2013)
69. XBMC Foundation: Third-party forks and derivatives. http://kodi.wiki/view/Third-party_forks_and_derivatives
70. Zhang, J., Chen, C., Ma, J., He, N., Ren, Y.: Usink: smartphone-based mobile sink for wireless sensor networks. In: Proceedings of the CCNC 2011 (2011)

Information Systems: Development, Research,
Applications, Education

9th SIGSAND/PLAIS EuroSymposium 2016, Gdansk,
Poland, September 29, 2016, Proceedings

Wrycza, S. (Ed.)

2016, X, 205 p. 35 illus., Softcover

ISBN: 978-3-319-46641-5