

Cross-Platform Development Tools for Smartphone Applications

Julian Ohrt and Volker Turau

Hamburg University of Technology, Germany



Developers can use cross-platform mobile development tools to create smartphone apps that meet user expectations, but existing XMTs need improvement.

The smartphone market has grown steadily in recent years, with unit shipments surpassing even those of PCs in late 2010.^{1,2} This growth has fueled demand for a wide range of new mobile applications by both consumers and businesses.

The rapid development of hardware and software platforms for smartphones by various vendors has resulted in a large number of mobile operating systems (MOSs). Currently, four MOSs—Symbian, Android, BlackBerry OS, and iOS—dominate the market,³ but at least five other, smaller MOSs have a dedicated user base: Windows Mobile (WinMob), Windows Phone 7 (WP7), bada, webOS, and MeeGo. Moreover, the market is highly dynamic. Research In Motion is moving from the BlackBerry OS to the QNX-based BlackBerry Tablet OS; Nokia will forego Symbian in favor of WP7; and Tizen, a new, Linux-based OS, will replace MeeGo.

The diverse and continually evolving MOS landscape constitutes a huge challenge for application developers. Unlike the desktop computer market, where more than 90 percent of users use Windows, mobile app developers must target multiple platforms to reach the same number of users. However, MOSs come with their own software development kit (SDK), each of which uses a unique programming language,⁴ and provide their own custom API. In short, developers must write an application separately for each mobile platform. Mobile app providers thus face

a dilemma: either invest considerable resources developing the same app for all mobile platforms, or leave some platforms unsupported and risk alienating potential customers.

Mashup⁵ services like Andromo App Maker (www.andromo.com), iBuildApp (www.ibuildapp.com), and AppMakr (www.appmakr.com) make it possible to create smartphone applications using dialogs, without the need for native SDKs. However, their capabilities are usually limited to Internet-related resources, like embedding RSS feeds or Twitter channels. Other functionalities, such as accessing local databases or an address book, are generally not available. Further, these services do not support several platforms using the same configuration.

Anthony Wasserman⁶ outlined two options for reducing application development efforts while still supporting multiple mobile platforms: use Web browsers to create platform-independent apps, or use cross-platform mobile development tools (XMTs) to create apps for different smartphone platforms from the same code base.

We focus on the latter approach by providing an overview of current XMTs, comparing their advantages and drawbacks.

SMARTPHONE APPS

Three aspects of smartphone applications that are important for understanding XMTs are the way such apps

are installed, their internal structure, and their graphical user interface (GUI) elements.

Installation

Whereas users usually can get software for desktop computers from various sources—including CDs, DVDs, or the Internet—and afterward install it on any number of devices, they can only obtain smartphone apps via dedicated app stores. Each MOS provider operates its own app store that only offers applications for that MOS. After creating an application for a particular MOS, the developer submits it to the corresponding app store. Once the app store approves the application, all users of that MOS can download and install it.

For some MOSs like iOS or bada, the vendor's app store is the only way for a developer to market applications. An application banned from the app store cannot be distributed. Because each app store has its own rejection policies, potentially at least one store could reject every app. Although they primarily reject apps for illegal or violent content, app stores can reject apps for other reasons. For example, in 2010, Apple prohibited apps not "originally written in Objective-C, C, C++, or JavaScript" in its app store. App stores often reject applications that download executable code or interpret code not contained within the application archive.

XMTs thus must neither apply technologies nor include features that might cause an app store to ban created applications.

Application structure

The safest way for a developer to avoid rejection of an app for technological reasons is to use the SDK provided by the target MOS vendor. Application packages' structure and content differ greatly depending on the native SDK. For example, Android's SDK creates Dalvik bytecode, a derivative of Java bytecode optimized for resource-restrained devices; XCode for iOS compiles to binary code; and webOS apps consist mainly of HTML, JavaScript, and image files. Thus, "native app" has a distinct meaning for each MOS.

Integrated apps. While the native output format is SDK dependent, each SDK creates archives installable on smartphones that run the corresponding OS. These applications are integrated into the system after installation—that is, they appear in the default application list and can access system functionalities.

Integrated applications can be developed as native or interpreted apps. Both are well-known approaches for creating cross-platform software on desktops—for example, QT for natively running software and Java for interpreted programs. However, whereas Java programs are run by launching the Java runtime, which in turn invokes and executes the Java program, interpreted apps for MOSs are always launched directly. This implies that at least the entry

point of each MOS application contains native code. Interpreted applications then launch the appropriate runtime environment, which in turn interprets the program code, while native apps continue executing native code.

In the context of XMTs, we can identify four basic categories of mobile apps: (1) purely native applications that access the system API directly, (2) purely native applications that access the system via a library providing an abstraction level, (3) interpreted applications that include the VM in the application package itself, and (4) interpreted applications that require a VM installed as a separate application. For interpreted applications, we do not distinguish whether a runtime environment is a process virtual machine or an interpreter;⁷ we refer to either as a VM.

The safest way for a developer to avoid rejection of an app for technological reasons is to use the SDK provided by the target MOS vendor.

Nonintegrated apps. These include Java Midlets, Flash applications, or webpages containing application functionality (Web apps). In general, separate tools like a Web browser invoke these apps, and they do not allow direct access from the system's app list. Developers usually use third-party tools to create nonintegrated apps. Most of these tools do not allow access to smartphones' advanced features—for example, Java Midlets do not have multitouch functionality, and Flash applications cannot access Bluetooth hardware.

GUI elements

Mobile app GUIs consist of standard elements like buttons, labels, or list boxes. There are three main types of GUI elements.

Native. Typically, the most responsive GUI elements are those that the system itself uses and that the corresponding SDK supports.

Web-based. Alternatively, smartphone apps can rely on the system's Web browser to provide the GUI. In this case, the application embeds a Web view that renders HTML code and thus displays browser elements such as HTML input fields that developers can modify using Cascading Style Sheets (CSS). Using Web technologies, developers can achieve a uniform look and feel across platforms, provided that different browsers implement rendering identically. At the same time, OS-specific CSS files make it possible to imitate native GUIs.

Custom. The third option is to create custom GUI elements that the app itself draws and controls. This approach is the most flexible but also the most complex, easily

Table 1. XMT application structure and GUI elements.

XMT	Version	Application structure				GUI elements		
		Purely native		(Partly) interpreted		Native	Web-based	Custom
		(1)	(2)	(3)	(4)			
Flash Builder	4.5			✓	✓			✓
Illumination Software Creator	4.0	✓				✓		
LiveCode	4.6.4			✓				✓
Marmalade	5.1.5			✓				✓
MoSync	2.6		✓	✓		✓		✓
OpenPlug Studio	3.0.9			✓		✓		✓
PhoneGap	1.1.1				✓		✓	
RhoStudio	3.0.2			✓	✓	✓	✓	
Titanium	1.7.1			✓		✓		

prolonging development time and inhibiting application performance.

CROSS-PLATFORM MOBILE DEVELOPMENT TOOLS

Developers can use XMTs to create integrated applications for several smartphone platforms from the same code base that app stores will generally accept.

XMT taxonomy

Table 1 categorizes nine XMTs in terms of the created apps' internal structures and GUI elements. The four numbers in the "Application structure" columns correspond to the four categories of mobile apps. The GUI classification refers to the system's default approach: most XMTs that use native or custom GUI elements also can embed a Web view to display HTML elements. Each system creates apps for at least Android and iOS. We obtained as much data as possible on these XMTs from vendor documentation and supplemented this information with a subjective analysis of the compile process.

Flash Builder. Developers can use Adobe Flash Builder (www.adobe.com/products/flash-builder-family.html), a stand-alone SDK built on the Eclipse platform, to create interpreted applications using the ActionScript language or open source Flex app framework. Depending on the MOS, AIR runtime software is either included in the XMT package (captive runtime support) or must be installed on the target system. For example, iOS only allows captive runtime support because Apple's App Store does not allow apps to interpret code not contained within the app archive itself. For Android, Flash Builder supports captive runtime since AIR 3.0.

Illumination Software Creator. Radical Breeze's Illumination Software Creator (<http://radicalbreeze.com>) provides a drag-and-drop capability for use in designing standard GUI elements such as text boxes, buttons, and labels. Similarly, developers can access predefined, graphical blocks that represent events and apply other functions like basic mathematical operations and conditional statements. Unless defining custom blocks, developers need not write code. After finalizing an Android or iOS app, they can use the corresponding SDK to compile it.

LiveCode. Runtime Revolution's LiveCode (www.runrev.com) is both an XMT and a programming language. Compiling Android and iOS applications requires using the corresponding SDKs. Although LiveCode's compile process is not officially documented, newsgroup posts by developers indicate that it uses interpreted code, with the VM embedded in the application package. However, an analysis of Android apps created with LiveCode suggests that developers can also create purely native apps using integrated libraries.

Marmalade. The Marmalade SDK (www.madewithmarmalade.com) requires using either Microsoft Visual Studio or Apple XCode. Developers write the applications in C++. An emulator allows debugging. The compile process is not documented, but an analysis of compiled Android apps indicates that Marmalade uses interpreted code with an integrated VM.

MoSync. Like Flash Builder, MoSync (www.mosync.com) is a stand-alone SDK based on the Eclipse Platform. During the compile process, it first transforms C++ source code into platform-independent intermediate code and then creates the actual application package in a second step. Depending on the MOS, MoSync contains precompiled

libraries together with the intermediate code compiled to either MoSync bytecode or native code. In the case of bytecode, the app also includes a VM. Again depending on the target platform, the VM can contain an ahead-of-time compiler that compiles MoSync bytecode to native code on the smartphone.

OpenPlug Studio. No longer maintained as of 15 December 2011, OpenPlug Studio (<http://developer.openplug.com>) was available as an Eclipse plug-in as well as a stand-alone SDK based on the platform. Developers wrote program logic in ActionScript and defined the GUI in Adobe's MXML markup language. Although the compile process was not documented, analysis revealed that OpenPlug Studio compiled source code to a custom binary code and combined this with a precompiled library in the app package. Whereas the binary code appeared to be bytecode, the precompiled library most likely contained the corresponding VM.

PhoneGap. Writing Adobe PhoneGap (<http://phonegap.com>) apps requires using the native SDK for each targeted MOS. PhoneGap provides native source code or libraries for each supported platform. It also includes a template for creating native smartphone apps that is extended with HTML and JavaScript files. Developers use HTML code to define the GUI, which a full-screen Web view element renders. The Web view element interprets JavaScript code, which can access local hardware via an interface provided by PhoneGap.

RhoStudio. Rhomobile's RhoStudio (<http://docs.rhomobile.com/rhostudio/tutorial>) is an Eclipse plug-in for developing and debugging mobile applications using the open source Rhodes framework. Developers use Ruby to implement program logic. At compile time, RhoStudio converts the source code into bytecode and wraps it with a VM in the app package. The GUI is based on a Web-view element that shows webpages served by an embedded web-server. With the integrated Ruby interpreter, developers can use eRuby to create dynamic webpages.

Titanium. Appcelerator's Titanium (www.appcelerator.com) is a stand-alone SDK also built on the Eclipse platform. Developers write all source code in JavaScript. Using APIs offered by Titanium, they can access system functions as well as define the GUI. During compilation, Titanium combines source code with a JavaScript interpreter and other static content into an app package. At runtime, the interpreter processes JavaScript code.

XMT challenges

Considering the broad variety of concepts XMTs apply to create applications on multiple smartphone platforms, a simple comparison of XMTs across common categories is infeasible. However, XMTs' internal details are of secondary interest to developers as long as app stores accept created applications, and most end users are oblivious of the underlying technology. For practical purposes, then, the most important considerations are whether an XMT

satisfies developer needs and whether the resulting apps meet user expectations.

Developer needs. A major requirement for developers is the ability to target as many platforms as possible with the least amount of effort, which generally equates to a minimum of coding—preferably, one programming language within one development environment. However, working from a single code base presents several problems.

First, the developer must be able to implement concepts and services that are handled differently among smartphone platforms, including multithreading, push services from a server to the application, and in-app ads.⁸ The XMT must either shield the developer from these differences with an abstraction layer sufficiently generic to cover all platforms, or it must provide options to more easily manage concepts and services individually.

Even more problematic are functionalities not all platforms support or that are unique to individual platforms—for example, Live Tiles is only part of WP7. Most XMTs deal with this issue by offering the lowest-common-

The most important considerations are whether an XMT satisfies developer needs and whether the resulting apps meet user expectations.

denominator subset of features. Thus, before deciding whether to use an XMT, developers should carefully consider what functionalities they need and whether it provides them. Advanced features of target platforms might not be available.

In addition, developers prefer an XMT that provides identical and correct behavior for all target platforms, which theoretically limits debugging and testing. Unfortunately, no available XMTs provide such a guarantee. However, this problem is not unique to XMTs—it also arises for different versions of the same platform.

Ideally, an XMT would also support distribution of the final version of an application to various app stores. Other desirable functionalities can accelerate app development including compilation without the need for native SDKs, code completion and refactoring, a GUI designer, a debugger, an emulator, and profiling capabilities.

User expectations. End users expect smartphone apps to install quickly and to be functional and intuitive. It should also be possible to install and run applications in parallel. XMTs can influence these attributes to varying degrees.

XMTs can speed up app installation by creating small app packages. They can enhance responsiveness by providing nimble GUI control elements and a short launch time.

Table 2. XMT support of various mobile operating systems.

XMT	Mobile operating system								
	Android	bada	BlackBerry	iOS	MeeGo	Symbian	webOS	WP7	WinMob
Flash Builder	✓			✓		✓		✓	✓
Illumination Software Creator	✓			✓					
LiveCode	✓			✓					
Marmalade	✓	✓	✓	✓		✓	✓		
MoSync	✓		✓	✓	✓	✓			✓
OpenPlug Studio	✓		✓	✓		✓			✓
PhoneGap	✓	✓	✓	✓		✓	✓	✓	✓
RhoStudio	✓		✓	✓		✓		✓	✓
Titanium	✓			✓					✓

XMTs can facilitate concurrent usage by making applications more resource efficient—that is, by limiting persistent and working memory usage.

Intuitiveness, which is closely connected to design, is difficult to measure and thus hard to achieve. Users of a given platform are accustomed to apps following certain user interface and experience conventions—for example, Android phones have a compulsory hardware back button and thus do not require an application to provide such an input field. Apps that do not follow such conventions might seem alien to the point of being unusable. XMTs can provide platform-specific GUIs to optimally comply with user expectations.

XMT COMPARISON

To assess XMT effectiveness, we compared the nine XMTs listed in Table 1 according to several objectively measurable properties. In terms of developer needs, we focused on features and functionalities intended to reduce development effort. In terms of user expectations, we considered various performance characteristics. It should be emphasized that other XMTs are available on the market and thus this evaluation is not exhaustive.

Features and functionalities

Table 2 shows which XMTs support which MOSs. All of the tools we evaluated support Android and iOS. BlackBerry, Symbian, and WinMob are also well-supported. Few XMTs support recent or less popular MOSs like bada, MeeGo, webOS, and WP7. None of the tools support all MOSs.

Table 3 compares some of the features and functionalities these XMTs offer to help speed up the development process.

The option of using a familiar programming language can be a strong incentive to select a certain XMT. Illumina-

tion Software Creator uniquely requires no programming at all, but its expressiveness is limited to the capabilities of the provided drag-and-drop elements. Currently, this XMT does not offer access to hardware or system functionalities, and it relies on the target platforms' SDKs for compiling, debugging, and emulating.

Some XMTs allow direct compilation of apps without the need for the target platform's native SDK. This frees the developer to download, install, get acquainted with, and possibly register for several native SDKs.

Unfortunately, many XMTs do not provide features that facilitate programming such as code completion, a graphical GUI designer, or a debugger—all of which Android's and iPhone's SDKs support.

XMTs typically include an emulator for testing and debugging. In general, these are very fast—faster than Android simulators—and thus increase productivity. However, most emulators are VMs, and if they are not fully compatible with the VM implementations on the target platforms, problems might not be detectable before testing on the hardware devices.

Most XMTs that do not support certain system functionalities let developers extend functionality using native code. However, this is a complex task that requires writing, debugging, and testing native code for all targeted platforms. An example of hardware with poor XMT support is Bluetooth, possibly because no generally available system API exists. For example, WP7 provides no API at all, and iOS only allows Bluetooth connections between Apple handheld devices. Support for other hardware or system functionalities is considerably better. For example, with the exception of Illumination Software Creator, all XMTs provide an API to access the camera, and all except LiveCode provide access to the local contact list.

Table 3. XMT features and functionalities.

XMT	Version	Programming language	Compile without SDK	Code completion	GUI designer	Debugger	Emulator	Extensible with native code	Bluetooth support
Flash Builder	4.5	ActionScript and MXML	✓	✓	✓	✓	Own	✓	×
Illumination Software Creator	4.0	None (drag-and-drop)	×	×	✓	SDK	SDK	✓	×
LiveCode	4.6.4	LiveCode	×	×	✓	✓	Own	×, except iOS	×
Marmalade	5.1.5	C++	✓	✓	×	✓	Own	✓	×, except iOS
MoSync	2.6	C++	✓, except iOS	✓	×	✓	Own	✓	✓, except WP7
OpenPlug Studio	3.0.9	ActionScript and MXML	✓	×	×	×	Own	✓	×
PhoneGap	1.1.1	HTML and JavaScript	×	✓	×	×	SDK	✓	×
RhoStudio	3.0.2	Ruby	×	✓	×	✓	Own	✓	✓, except WP7 and Symbian
Titanium	1.7.1	JavaScript	×	✓	×	✓	SDK	✓	×

In general, there is no easy way to determine whether an XMT supports a specific system feature or functionality. Documentation can be out-of-date or even unavailable. Complete support of a functionality is often restricted to particular platforms.

Performance

Several performance attributes of applications created with XMTs are measurable. Small file size correlates to a fast installation time. A short launch time increases responsiveness. Less persistent memory enables installation of more apps in parallel; low RAM usage is essential to running apps concurrently.

To obtain comparable performance metrics, we created a simple application on all nine XMTs that consisted of one screen, which shows a small text label, and a default icon. We compiled each app for Android in release mode. File size was the first data point. We determined launch time and memory usage during application execution on an Android 2.2 simulator. To confirm that performance was similar on a real smartphone, we also executed the sample apps on a low-end Huawei Ideos X3 device running Android 2.3.3.

Some XMTs create apps that install themselves on first launch. For these tools, we only considered repeated launches. We measured launch time from starting the application until the text label appeared. To determine required RAM, we used the procrank utility to measure the running app’s unique set size (USS)—the amount of memory

freed on closing the application—via the Android emulator’s debugging shell. Because procrank is not accessible on real smartphones, we give only the values measured on the emulator.

Unfortunately, we could not launch the evaluation app created with Flash Builder on either the emulator or on the Huawei smartphone because we could not install the VM. In this case, we used the more powerful HTC Desire smartphone, on which we could download the 8-Mbyte Adobe AIR package from Android Market. After this installation, which required 24 Mbytes of persistent memory, we successfully launched the evaluation app.

The right half of the upper part of Figure 1 shows the launch time of the evaluation applications created with the nine XMTs, sorted from fastest at the top to slowest at the bottom. For reference, the first row contains results obtained running a native Android implementation of the app. For comparison, the bottom right of the figure shows the launch time of Qype, a purely native Android application that lets users list, comment on, and upload photos of places and companies of public interest. Qype provides considerably higher functionality and a more complex GUI than the evaluation app. Of its file size, 1.5 Mbytes consist of static resources like images.

The left half of the upper part of Figure 1 shows the apps’ size and required RAM, with the corresponding Qype values below.

The results clearly demonstrate that the choice of XMT strongly influences overall app performance. Only four

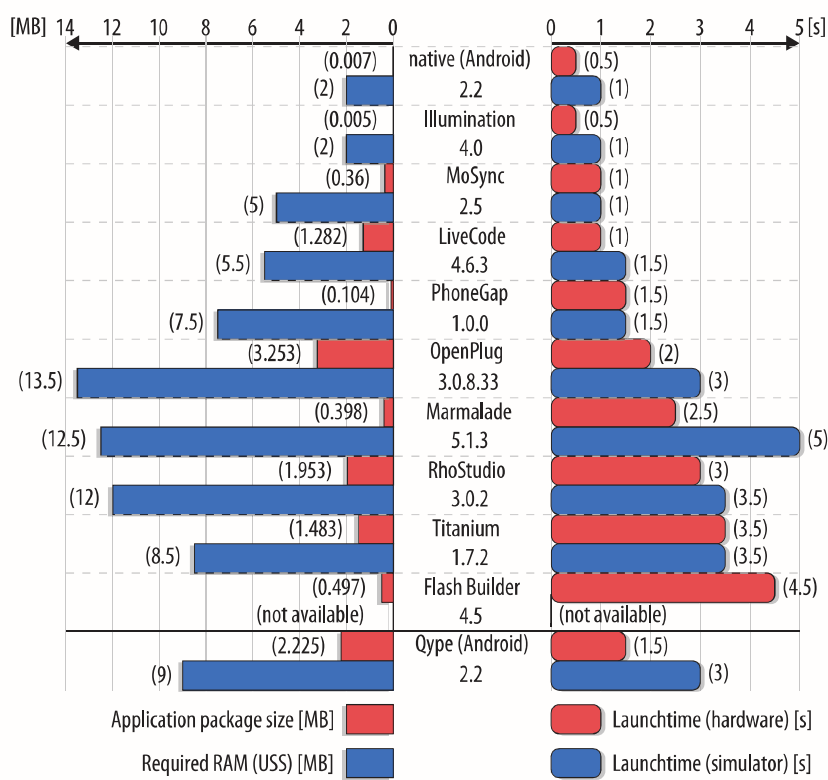


Figure 1. Performance characteristics of a simple evaluation application created on the nine XMTs, along with those of the native Qype Android app. The choice of XMT strongly influences application performance.

of the XMTs—Illumination Software Creator, LiveCode, MoSync, and PhoneGap—created evaluation apps that launched at least as fast as and required less RAM than Qype. This means that, alone, the VMs deployed by the other XMTs consumed more resources than a full-grown, purely native application.

Future evaluations should consider XMTs' impact on runtime performance, as well as how launch time and memory usage scales for more complex apps. Comparing applications on other platforms, such as iOS, would also be valuable.

The future of XMTs strongly depends on the future of mobile operating systems. The greater diversity of MOSs and the more equal their market share, the more important XMTs will become. However, as MOSs are free-market products, their future cannot be reliably predicted. Past experience shows that a shakeout is not the only option:

- Of three potential successors to the DVD optical storage medium—Blu-ray Disc, HD DVD, and VMD (Versatile Multilayer Disc)—only Blu-ray technology is used today.

- Windows dominates the desktop computer market, but Mac and Linux, despite having very small market shares, continue to exist.
- The home videogame console market is roughly divided equally among three devices: PlayStation 3, Wii, and Xbox 360.

Market research firms Gartner³ and IDC⁹ both predict a distribution of MOSs in 2015 similar to those of videogame consoles: Android is expected to have 46 percent of the market, followed by WP7 (20 percent), iOS (17 percent), BlackBerry (12 percent), and all others (5 percent). If these predictions hold, XMTs will play an important role in smartphone application development.

However, XMTs must improve considerably to become a serious alternative to native SDKs for two reasons. First, many current XMTs provide far less development support. Second, available XMTs create apps with much higher resource requirements than purely native applications. Further, XMTs must offer better support for platform-specific user interface and user experience requirements.

Where hiding device fragmentation is infeasible or impossible, or would hide a platform's unique character, XMTs should help developers manage fragmentation.

Today's XMTs are merely suited for developing cross-platform applications, which impose modest demands on CPU power and memory resources, do not rely on cutting-edge technologies, and do not make high GUI demands.

It is also possible that mobile apps will follow the same trend that has characterized the PC market for the past several years, and Web apps will supplant native apps. In this case, neither native SDKs nor XMTs would be required. Instead, there would be a need for tools to develop applications that run within mobile Web browsers and make full use of the functionalities of the forthcoming HTML5 standard. At the same time, researchers would have to address other issues including interrupted or unavailable mobile Internet connections, access to system functionalities from within Web apps, and Web browser compatibility.

An advantage of moving to Web applications is that they do not need to be installed, and updates can be transparently and instantly made available to all users worldwide.¹⁰ This would weaken the position of app stores and deprive MOS vendors of full control of smartphone apps. Another argument favoring Web applications is that attempts to prematurely replace existing protocols and techniques with

new ones often fail. Thus, integrated mobile apps and app stores could share the same fate as the Wireless Application Protocol. ■

Acknowledgment

The German Federal Ministry of Education and Research, under funding code 03CL26B, supported the research described in this article.

References

1. Int'l Data Corp., "Android Rises, Symbian^3 and Windows Phone 7 Launch as Worldwide Smartphone Shipments Increase 87.2% Year Over Year, According to IDC," 7 Feb. 2011; www.idc.com/getdoc.jsp?containerId=prUS22689111.
2. Int'l Data Corp., "PC Market Records Modest Gains during Fourth Quarter of 2010, According to IDC," 12 Jan. 2011; www.idc.com/getdoc.jsp?containerId=prUS22653511.
3. Gartner, "Gartner Says Android to Command Nearly Half of Worldwide Smartphone Operating System Market by Year-End 2012," 7 Apr. 2011; www.gartner.com/it/page.jsp?id=1622614.
4. A. Charland and B. LeRoux, "Mobile Application Development: Web vs. Native," *Queue*, Apr. 2011, pp. 20-28.
5. F. Daniel, M. Matera, and M. Weiss, "Next in Mashup Development: User-Created Apps on the Web," *IT Professional*, Sept./Oct. 2011, pp. 22-29.
6. A.I. Wasserman, "Software Engineering Issues for Mobile Application Development," *Proc. FSE/SDP Workshop Future of Software Eng. Research (FoSER 10)*, ACM, 2010, pp. 397-400.
7. J.E. Smith and R. Nair, "The Architecture of Virtual Machines," *Computer*, May 2005, pp. 32-38.
8. R. Virkus et al., *Don't Panic: Mobile Developer's Guide to the Galaxy*, 9th ed., Enough Software, 2011; www.enough.de/fileadmin/uploads/dev_guide_pdfs/Guide_9thEdition_WEB.pdf.
9. Int'l Data Corp., "Worldwide Smartphone Market Expected to Grow 55% in 2011 and Approach Shipments of One Billion in 2015, According to IDC," 9 June 2011; www.idc.com/getdoc.jsp?containerId=prUS22871611.
10. T. Mikkonen and A. Taivalasaari, "Reports of the Web's Death Are Greatly Exaggerated," *Computer*, May 2011, pp. 30-36.


Julian Ohrt is a PhD student in the Institute of Telematics at Hamburg University of Technology, Germany. His research interests include indoor-location-based services for smartphones. Ohrt received an MSc in computer science from Hamburg University of Technology. Contact him at julian.ohrt@tu-harburg.de.

Volker Turau is a professor of distributed systems, and heads the Institute of Telematics, at Hamburg University of Technology. His research interests include wireless communication and energy reduction in distributed systems. Turau received a PhD in mathematics from the Johannes Gutenberg University of Mainz, Germany. He is a member of IEEE. Contact him at turau@tu-harburg.de.

cn Selected CS articles and columns are available for free at <http://ComputingNow.computer.org>.

IEEE TRANSACTIONS ON AFFECTIVE COMPUTING

A publication of the IEEE Computer Society



Affective Computing is the field of study concerned with understanding, recognizing and utilizing human emotions in the design of computational systems. The *IEEE Transactions on Affective Computing (TAC)* is intended to be a cross disciplinary and international archive journal aimed at disseminating results of research on the design of systems that can recognize, interpret, and simulate human emotions and related affective phenomena.

Subscribe today or submit your manuscript at:
www.computer.org/tac