Making Apps Useable on Multiple Different Mobile Platforms: On Interoperability for Business Application Development on Smartphones

Andreas Holzinger[1], Peter Treitler[1], and Wolfgang Slany[2]

[1] Medical University Graz, A-8036 Graz, Austria
Institute for Medical Informatics, Statistics & Documentation (IMI)
Research Unit Human-Computer Interaction
{a.holzinger,p.treitler}@hci4all.at
[2] Graz University of Technology, A-8010 Graz, Austria
Institute for Software Technology (IST)
wolfgang.slany@tugraz.at

Abstract. The relevance of enabling mobile access to business enterprise information systems for experts working in the field has grown significantly in the last years due to the increasing availability of smartphones; the shipment of smartphones exceeded that of personal computers in 2011. However, the screen sizes and display resolutions of different devices vary to a large degree, along with different aspect ratios and the complexity of mobile tasks. These obstacles are a major challenge for software developers, especially when they try to reach the largest possible audience and develop for multiple mobile platforms or device types. On the other side, the end users' expectations regarding the usability of the applications are increasing. Consequently, for a successful mobile application the user interface needs to be well-designed, thus justifying research to overcome these obstacles. In this paper, we report on experiences during an industrial project on building user interfaces for database access to a business enterprise information system for professionals in the field. We discuss our systematic analysis of standards and conventions for design of user interfaces for various mobile platforms, as well as scaling methods operational on different physical screen sizes. The interoperability of different systems, including HTML5, Java and .NET is also within the focus of this work.

Keywords: Mobile computing, user interfaces, smartphones, mobile platforms, app development, usability engineering, multi-platform, cross-platform.

1 Introduction and Motivation for Research

Mobile devices are gaining more and more importance and smartphones are nowadays a common sight in industrial countries. What started originally as a concept product by IBM in 1992, was a niche product for a long time. This changed rapidly with the release of the Apple iPhone in 2007, bringing smartphones to the mainstream. One year later, Google and the Open Handset Alliance released their Android platform, which gained a lot of popularity in the following years.

G. Quirchmayr et al. (Eds.): CD-ARES 2012, LNCS 7465, pp. 176–189, 2012.

As of June 2012, there were a total of more than 300,000,000 Android devices shipped worldwide, with more than 900,000 new ones being activated every day. Similarly, the iPhone is very popular with more than 180,000,000 devices shipped in total. The iPad, released in 2010, has been sold more than 55,000,000 times.

Along with this new and big market, new challenges for software developers emerge. Different programming languages, IDEs and platform standards need to be learned rapidly in order to develop native apps for multiple platforms.

HTML5 promises easy multi-platform development, with browsers on most current mobile devices being able to render HTML5 content. Therefore, ideally only a single app would need to be created which then could be executed on multiple target platforms.

However, there are also some disadvantages of HTML5, when compared to native apps, such as limited access to the device hardware or to platform features, look and feel that users are not used to from other apps of the platform, as well as missing market presence. In this paper we try to pinpoint the advantages and disadvantages of HTML5 apps and native apps and try to offer some guidelines as to which to choose for app development.

Besides the number of different platforms, mobile devices come with screens in different sizes, resolutions, and aspect ratios. Additionally, tablets are typically used in landscape mode, whereas smartphones are used in portrait mode, though both can also be used in the other way. This proliferation of interface properties holds especially true for Android, which supports devices with a display diagonal reaching from 2 inches over tablets which typically measure 10 inches up to TV screens with a display diagonal greater than 40 inches.

Many problems arise during the development of mobile applications – especially in industrial, professional and safety-critical environments [1], reaching from security aspects [2] to issues of the user interface [3]. In this paper we concentrate in on the latter and deal with questions including: Are different UI layouts required for different screen sizes? If so, where should designers draw the line between small and large devices? Are there tools which support designers in creating UIs for multiple screen sizes?

2 Background and Related Work

2.1 Business Case: Mobile Access to a Business Information System

During the research for this paper a project was carried out in cooperation with the Austrian business software development company Boom Software AG. The project was concerned with usability evaluation and consulting for the development of a mobile application. We will not discuss the details of the project here, but will illustrate some of the problems and challenges encountered in order to underline the relevance of mobile devices for business applications and the importance of usability of mobile apps. The information on the company and its technologies presented here is based on their marketing documents (January 2012) and personal communication:

Boom Software AG was founded in 1995 and has 50 employees (March 2012). The company primarily develops industrial software for maintenance, management

and production control. Boom Software advertises *total customization* [4] of its software in order to meet the specific needs and requirements of individual customers.

The underlying technology which enables the customization of their software is the Business Oriented Rapid Adaption (BORA) Framework. The BORA Framework was created to provide a unified design framework for the developers. According to company representatives, prior to the framework's introduction, their developers spent roughly 80% of their efforts on solving technical issues, whilst only 20% were used to customize the software to optimally meet the end users' needs. The framework provides the base technology for all software products and includes a number of tried and tested user interface elements and concepts, therefore drastically reducing the workload on technical tasks and allowing greater focus on customization.

Boom Software's products manage large amounts of enterprise data. The users, however, are not interested in just having access to the data; they need effective ways to efficiently gain knowledge out of this data [5]. A well-designed, usable interface may help to achieve that goal.

While the main focus of Boom Software's products has historically been on desktop software for Microsoft Windows systems, a platform-independent HTML-based UI layer for BORA applications has also been developed over the last few years. Recently, development on the first mobile app has started. Platforms targeted are Android and Windows 8 (for tablet PCs). Boom Software's goal is to make specific parts of software systems run on smartphones and tablets in order to increase the productivity of outdoor staff like maintenance workers.

2.2 Mobile Work

Many companies today rely heavily on their software systems in their everyday work. At the same time, many jobs require employees to be mobile rather than working in their office. Maintenance workers, for instance, need to be on-site to perform maintenance while they may also need to be able to access information in the company's databases.

This mobile work [6] has often been done on laptop computers where appropriate display resolution supported the necessary visual performance [7]. In the last years, many of these laptop computers were replaced by tablet computers [8]. Mobile work usually includes three dimensions: mobility, location (in-)dependency, and time criticality; and context-related mobile work support functions are: location tracking, navigation, notification, and online job dispatching. According to the task-technology fit (TTF) theory [9], [10] and attitude/behavioral theory [11], it is crucial to find a fit between task characteristics and mobile work support functions; guidelines for development and use of mobile work support systems can be found in [12].

2.3 Variety of Mobile Devices

One major challenge when developing mobile apps (for smartphones, tablets or both) is the wide range of different platforms and device types available.

As detailed in Table 1, there are currently six different mobile platforms, which had a significant market share in 2011: Android, iOS, Symbian, BlackBerry, Bada and Windows Phone.

Table 1. Smartphone shipments (in millions) and shares in the year 2011 by platform [13]

Platform	Full year 2011 shipments	Share	Growth Q4'11 / Q4'10
Total	487.7	100.0%	62.7%
Android	237.8	48.8%	244.1%
iOS	93.1	19.1%	96.0%
Symbian	80.1	16.4%	-29.1%
BackBerry	51.4	10.5%	5.0%
Bada	13.2	2.7%	183.1%
Windows Phone	6.8	1.4%	-43.3%
Others	5.4	1.1%	14.4%

2.4 Multi-platform Development

Developing applications that can be used on all these platforms is often extremely difficult. All platforms have different Software Development Kits (SDKs) along with different libraries and different ways to design user interfaces. The programming languages of these platforms differ as well (see Table 2).

Table 2. Programming languages on mobile platforms

Platform	Programming languages
Android	Java
iOS	Objective-C
Symbian	C++
BlackBerry	Java
bada	C++
Windows Phone	C#

Knowledge of the programming languages required for app development is, however, only a small part of the expertise needed. The SDKs, platform standards and best practices should be known to developers as well. On the legal side, the terms of use of the different markets can also vary to a large degree, e.g. both Apple and Microsoft ban software that contains parts licensed under any GNU license. Gaining all this knowledge for a large number of platforms is a very time-consuming task.

In addition to the platforms themselves, developers need to consider the devices available for these platforms. Android, for instance, has many different devices available, including small smartphones and large smartphones and tablets. IOS, likewise, runs on the iPhone and the iPad.

These devices have different screen sizes and sometimes different aspect ratios as well, which requires effort to make user interfaces well scalable – or sometimes the design of different user interfaces for different screen sizes altogether.

Chae and Kim (2004) [14] have found that the screen size of a device significantly impacts users' behaviors. They have also found different levels of breadth and depth of information structures to be more appealing to users for different screen sizes.

Web technologies, such as HTML5, CSS3 and JavaScript promise an easy solution to the platform segmentation on the mobile market today: Every platform includes a browser which can display web sites and web apps.

The HTML5 standard is still under development. This implies that the browsers used must be up-to-date in order to support all of the most recent features of HTML5.

Web apps have a number of advantages and disadvantages when compared to native apps. These will be addressed in the upcoming sections.

A third approach is the development of so-called hybrid apps. These are native apps which display a web user interface, which can thus be the same across platforms. The platform SDKs offer UI elements which can be used for this purpose.

2.5 Related Work

Work on solving the problem of rendering on multiple user interfaces began a while ago, however, to date very few work is available for mobile devices explicitly. Most of the work was created before the mobile platforms discussed became available and therefore don't consider these platforms and their capabilities specifically. In the following we determine general work versus explicit work on mobile user interfaces:

Farooq Ali et al (2001) [15] discuss the need for a unified language for describing user interfaces in order to facilitate multi-platform UIs. Their approach uses the User Interface Markup Language (UIML) to create a high-level logical model of the UI and then derive models for families of devices from it in a developer-guided process.

Stocq & Vanderdonckt (2004) [16] describe a wizard tool which is able to create UIs for simple and complex tasks based on the domain model of the information system. Their approach also requires input and choices made by the developers. The wizard can create UIs for the Microsoft Windows and PocketPC platforms. The mobile UI needs to be built separately, but some of the information from the creation of the desktop UI can be reused to reduce the workload for the developer.

The work by Mori et al. (2004) [17] focuses on *nomadic applications* which can be accessed by the user on a variety of different devices and platforms as well as the different contexts of use of the applications on these platforms. Their *One Model, Many Interfaces* approach attempts to avoid low-level details by the use of meaningful abstractions. One central task model is created for the nomadic application. System task models, abstract and concrete UIs are then derived incrementally. They also stress that developers need to be aware of the potential target platforms early in the design process in order to create suitable tasks for each one.

Choi et al. (2009) [18] describe an application framework based on the Model-View-Controller pattern to assist the agile development of multi-platform mobile applications. Their approach uses a set of rules for each target platform in order to transform the UI.

3 Methods and Materials

In our research we have closely examined the Android and iOS platforms, along with the tools they offer to design and scale user interfaces and the guidelines they offer on UI design. The issue of web apps versus native apps was also thoroughly surveyed.

3.1 Android

Android is currently available on a wide range of devices. Device manufactures must adhere to Google's Compatibility Definition Document (CDD). According to the current version of the CDD [19], the physical display diagonal of devices must be at least 2.5 inches. The minimum display resolution must be 320 x 240 pixels (QVGA). The aspect ratio must be between 4:3 and 16:9.

The Android framework defines four different classes of screen sizes. These are small, normal, large and extra-large. In addition, there are four density classes: Ldpi (120 dpi), mdpi (160 dpi), hdpi (240 dpi) and xhdpi (320 dpi). Devices should report the next lowest size and density standard, meaning sizes and densities can be anything between and above the standard values.

A detailed breakdown of device shares per size and density can be found in Table 3. This data shows that more than 80% of all Android devices fall into the normal size category.

Table 3. Shares of Android devices per size and density [20]. Data collected by Google and based on devices which accessed the Google Play store in early March 2012.

	ldpi	**mdpi**	**hdpi**	**xhdpi**
small	1.7%		2.4%	
normal	0.7%	18.5%	66.3%	2.5%
large	0.2%	2.8%		
xlarge		4.9%		

3.2 iOS

Unlike Android, iOS only runs on a few devices manufactured by Apple. These include the iPhone, the iPod Touch and the iPad (all in different versions). Furthermore, iOS is the operating system for Apple TV, but this is outside the focus of this paper and will not be discussed in detail.

With regard to screen sizes, this means that developers must support two different sizes, as the iPhone and iPod Touch screens are identical in size. Figure 1 shows the size difference of the devices while Table 4 gives a detailed overview of the screen specifications of mobile iOS devices.

Fig. 1. A side-by-side comparison of the mobile iOS devices. Individual image sources: Technical device specifications as found on www.apple.com.

Table 4. Comparison of the displays of iOS devices. Source: Technical device specifications as found on www.apple.com.

Device	Screen size	Resolution	Aspect ratio
iPhone (up to 3G)	89mm	480 x 320	3:2
iPhone (4 and 4S)	89mm	960 x 640	3:2
iPod Touch (up to 3rd gen.)	89mm	480 x 320	3:2
iPod Touch (4th gen.)	89mm	960 x 640	3:2
iPad (up to 2nd gen.)	250mm	1024 x 768	4:3
iPad (3rd gen.)	250mm	2048 x 1536	4:3

As can be seen in Table 4, all iOS devices had their resolution doubled in a newer generation at some point. In order to make it easy to deal with these differences, Apple has introduced a measurement unit called points. Points, much like Android's density-independent pixels, server to define layouts and measurements independently from display density. One point, for instance, equals one physical pixel on the first iPhone generation while it equals two physical pixels on the iPhone 4S.

Image resources can (and should) also be provided in two different resolutions. The proper image will then be selected and displayed on the device the app is used on.

Beside these simple tools, iOS developers can be sure that the device market will remain manageable in the future with Apple being the only device manufacturer.

3.3 Multi-platform Development

The question whether to develop a native app or a web-based HTML5 app is one which every app developer will inevitably ask herself or himself. The different platforms and environments of mobile devices (see Table 2) make porting an app from one platform to another one a difficult task.

HTML5 is a promising alternative - every one of these platforms has a browser and the capability to run web apps. While HTML5 apps can run on any mobile platform with little to no extra effort, native apps also have advantages (see Table 5). Access to the system's hardware or some hardware-related features such as the camera is often only possible in native apps. Native apps can also integrate themselves seamlessly into the system: They use the native look-and-feel, can offer widgets to be displayed on the home screen, interaction with other apps, running as services in the background, and the use of system notifications. Native apps also tend to have better performance than web apps, though current browsers are getting more and more performance optimizations while at the same time newer devices offer more computational power, making the difference diminish.

Table 5. Advantages of native and HTML5 web apps. Sources: Meier and Mahemoff (2011) and own original research

Native app advantages	Web app advantages
• Broad access to device hardware and platform features • Close integration with platform and other apps • Better performance • Visibility through app store(s) • On-device storage	• Large target audience (mobile platforms + desktop) • Easy multi-platform development • Easy to update across all platforms • No certification required

It is also possible to develop hybrid applications. These are native applications which use views to display HTML5 content, which can be the same across all platforms.

3.4 Multi-platform Frameworks

Since the development of apps for mobile platforms is getting more important with numerous platforms being available on the market, platforms have been developed which should facilitate easy app development on multiple platforms. These multi-platform frameworks come in many variations. Some of them only change the visual appearance of web apps while others offer whole software development kits on their own, which can build the developed apps to multiple target platforms as native apps.

For this paper, we have briefly examined three frameworks: JQuery Mobile, PhoneGap and Titanium Mobile. There are, of course, many more frameworks available. Criteria for the selection of the frameworks were that these were, at the time, some of the most feature-rich and most popular. The aim was also to have

frameworks with different key features in order to be able to compare them. Furthermore, the frameworks are also available for free. There are several proprietary frameworks but they could not be tried out thoroughly and were therefore excluded from the selection.

3.4.1 JQuery Mobile

JQuery Mobile [21] is a web app framework which was developed by the developers of the popular JavaScript library jQuery. The main focus of jQuery Mobile lies on the design and presentation of mobile apps.

The technologies used by jQuery mobile are HTML5, CSS3 and JavaScript, which enables it to run on most current mobile devices. JQuery Mobile can be used to both make web apps feel more like native apps and to apply a brand layout to them, giving them a more polished look than a default web app. The layout is scaled to the target device. Small adaptations can be made automatically. For instance, labels can be placed beside input fields on larger devices while being placed above them on smaller devices.

The jQuery mobile website offers a drag-and-drop tool for building simple UI layouts called Codiqa. Components such as buttons, text, images and lists can be dragged into the UI, allowing for rapid prototyping. The resulting source code can then be downloaded and further modified. This makes it much easier for new or inexperienced developers to get started, as the other platforms such as Android have a much steeper learning curve.

The jQuery Mobile framework is free and open source. It can be used under the terms of either the MIT License or the GNU General Public License. The basic version of the Codiqa, which allows the download of the source code created, is free. Advanced versions which offer additional features cost 10$ per month for single users and 30$ per month for teams.

JQuery Mobile offers some more features which are not strictly related to design, most notably the handling of events such as various forms of touch input (e.g., tap, swipe) and orientation change (i.e., the user turning the device from portrait to landscape mode or vice versa).

3.4.2 PhoneGap

PhoneGap [22] is an app platform which can build apps from a single code base for multiple target platforms as native apps. It is based on HTML5, CSS3 and JavaScript and abstracts the hardware of mobile platforms into a JavaScript API. The library provides interfaces for hardware access and certain native features. Hardware access includes the device's accelerometer, the camera and the compass.

Some of the platform features are access to the contact list or to the file system, notifications and geolocation. All of these features are available for Android, iPhone (3GS and above) and Windows Phone 7. Other platforms such as Blackberry or Symbian have a subset of these features available. PhoneGap thus manages to combine many of the advantages of native apps and web apps.

Technically, the resulting apps generated by PhoneGap are not truly native apps but hybrid apps. The apps use web views which display the HTML5 content, but they can be built to the platform's native format and therefore also distributed over several app stores.

Developers need to set up the SDKs for all platforms they want to develop for. This implies some limitations: In order to deploy build PhoneGap apps for iOS the iOS SDK is required, which in turn requires a Mac OS computer. PhoneGap supports the default IDEs for various platform (e.g., Eclipse with the Android Development Tools for Android) and only requires the libraries to be imported.

PhoneGap is an open source project and is available for free. It uses the Apache License (version 2.0).

3.4.3 Titanium Mobile SDK

The Titanium Mobile SDK by Appcelerator Inc. also offers the possibility to create native apps from a single code base. It comes with a complete IDE called Titanium Studio, which is based on the Aptana Studio web development UI. The app logic is written using JavaScript.

Similar to PhoneGap, Titanium offers an extensive API with access to native hardware and software features. It allows the use of additional JavaScript libraries such as jQuery. It currently supports iOS, Android and BlackBerry and also allows the creation of mobile web apps (the latter still being in beta status).

Appcelerator also offers a marketplace for templates, designs and modules for applications.

In contrast to PhoneGap, Titanium Mobile UIs are using real native UI components rather than web views. Like PhoneGap, Titanium Mobile requires the SDKs of the target platforms to be installed and set up within the SDK. It uses the Apache License (version 2.0), but the software is proprietary. The basic version of Titanium Mobile is free. There is also a Professional Edition and an Enterprise Edition. These offer additional support, some additional modules and early access to new features.

The feature of creating native UIs for different platforms is very promising. The SDK was only briefly examined for this paper and further investigation is necessary in order to evaluate how well Titanium handles the rendering of complex native UIs.

While the building of native apps can be an advantage, it also means more overhead as a number of different apps needs to be built and tested with every new update.

3.5 Testing User Interfaces

Testing mobile user interfaces for different screen resolutions in itself is already a major hassle, and testing multi-platform mobile user interfaces can be even much worse [23], [24], [25]. Manual testing is practically feasible only for the most simplistic applications because typical software is usually far too complex to be tested thoroughly, and is continuously developed as it is changed or extended with new functionality.

Automatic testing can be a solution but the test code needs to be continuously kept up-to-date, and the problems arising with different screen sizes and systems are much intensified as then all tests need to be constantly maintained for all systems and display resolutions.

Mesbah & Deursen (2009) [26] propose a method for testing AJAX applications automatically by introducing CRAWLJAX, a crawler for AJAX applications, which can dynamically make a full pass over an AJAX application.

However, the medium term benefits of testing, especially when using test driven development style automatic GUI tests, are huge, e.g., in terms of developer

satisfaction due to trustable software documentation (tests can be run and checked – they serve as minimalistic declarative specification/programmer documentation that can be consulted and really correspond to the actual code) and reduced risks associated with evolving code, in particular when refactoring the code of other developers in larger teams (see [27] for more details).

A further problem that is particular to the Android platform is that different hardware providers often add their own UI customization that can introduce subtle changes in the behavior of otherwise identical Android version. E.g., HTC Sense substantially changes the behavior of certain buttons, fields, and the keyboard, thus necessitating either to deal with these differences on an individual basis in automatic GUI tests, or to ignore such hardware dependent differences, in particular as these customizations (which can be quite popular as in the case of HTC or Samsung Android devices) are not available on, e.g., the official Google Android emulator. This of course entails either to write tests for all these differences and to test on actual hardware, which needs a rather complicated setup for automatic tests, or to not test for them, which of course is also far from ideal.

The problem is further complicated for all platforms by the different OS versions. For example, most Android phones currently run Android version 2.x. New models are delivered with Android version 4.x, and older models are increasingly updated to this newer version. However, fundamentally new GUI possibilities have been introduced in the newer versions, e.g., the possibility to have several resizable but non-overlapping "windows" on the screen at the same time (with some limitations), or to have resizable "gadgets" on the home-screen. Testing for different OS versions thus can further complicate the issues for developers, though being able to run automatic GUI tests on an emulator that is able to execute different OS versions can be very helpful. Note however that, e.g., Google's Android emulator does not support OpenGL 2.x and thus is of limited value for game developers. It unfortunately is also very slow.

Additional problems arise when one wants to test hardware features such as sensors (e.g., GPS or gyro-sensors where we would like to simulate sensor input for testing purposes – the OpenIntents sensor simulation framework solves this at least for Android), effectors (e.g., the flash light or the vibration feature), or connectivity like Bluetooth protocol stacks and internet access.

Nevertheless, there exist some very useful tools that allow automatic GUI testing of mobile apps for Android, iOS, Windows Phone devices as well as HTML5/Javascript based applications, including, e.g., Robotium and Roboelectric for Android, or Selenium 2 for mobile HTML5/Javascript (Android and iOS are particularly supported).

4 Results

Our research has shown that different platforms allow different ranges of screen sizes, with the Android platform supporting the highest variability along with a wide collection of tools for designing UIs for multiple screens, such as density independent pixels and size standards.

The iOS platform supports a strictly limited number of screen types. It handles the different resolutions of older and newer devices very well by measuring the resolution in the abstract unit of points rather than physical pixels. The same UI layout can therefore be used for old and new iPhones and iPod Touch devices with little regard to the different physical screen resolutions.

The iPad, however, differs from the other iOS devices in size, resolution and aspect ratio. It is therefore recommended to design separate UIs for iPad and iPhone / iPod Touch.

We have also observed that the platform SDKs offer ways to separate the UI definition from the application logic, therefore supporting the use of the Model-View-Controller (MVC) pattern [28]. We encourage developers to clearly separate the UI definitions from the rest of a code base. This separation makes it much easier to create scalable user interfaces.

Following platform conventions is very important on mobile devices, especially when developing native apps. Standard usability engineering methods [29] are important, but UI designers need to be aware of the platform-specific guidelines for all target platforms in order to successfully apply usability engineering principles to mobile applications [30].

When comparing native apps to web apps in general, there is no clear recommendation to be given. The choice of whether to develop native apps, web apps, hybrid apps or to use a multi-platform framework depends on a number of things. The requirements of the app and the resources available as well as the target audience should be considered.

The skills of the developers are another important factor, as learning how to develop apps for one or more mobile platforms can be very time consuming. It can be a great asset for app development to have developers who are already skilled in one or more platforms' native programming languages and APIs, or in HTML5, CSS and JavaScript. This advantage should be leveraged.

It is also worth mentioning that multi-platform frameworks are growing very rapidly and both updates to existing frameworks and the introduction of new frameworks occur frequently. We recommend comparing the features of the most recent versions of various frameworks before deciding on one.

Another very important thing is to be aware of the limitations of the frameworks. First and foremost, these frameworks only offer tools for creating UIs for multiple platforms and screen sizes. It is the responsibility of the designers and the developers to ensure that the apps run properly on all the target platforms and devices. None of the frameworks can substitute thorough testing of the app.

5 Conclusion

There are a number of different tools and means to scale user interfaces on the mobile platforms examined. These are, however, only tools and it is up to the designers and the developers to ensure that their user interfaces look good and are well usable on all target devices. While the well thought-out definition of layouts is a good foundation for this, it must be stressed that there is no way around testing the interface. This can be done on actual devices or on various configurations of the emulators provided by the platform IDEs, but there are many hard challenges associated with automatic GUI testing.

With regard to multi-platform development and the choice between developing native apps or HTML5 web apps, no definitive recommendation can be given. The best choice always depends on a number of factors, such as the app's intended features, the target audience or the skills of the development team.

Multi-platform frameworks offer some interesting features, especially for rapid prototyping or the development of very simple apps. These frameworks are growing rapidly and new features are being added frequently, so we recommend inspecting several platforms' features before starting multi-platform development and carefully choosing a well-suited framework.

References

1. Holzinger, A., Hoeller, M., Bloice, M., Urlesberger, B.: Typical Problems with developing mobile applications for health care: Some lessons learned from developing user-centered mobile applications in a hospital environment. In: Filipe, J., Marca, D.A., Shishkov, B., van Sinderen, M. (eds.) International Conference on E-Business (ICE-B 2008), pp. 235–240. IEEE (2008)
2. Weippl, E., Holzinger, A., Tjoa, A.M.: Security aspects of ubiquitous computing in health care. Elektrotechnik & Informationstechnik, E&I 123(4), 156–162 (2006)
3. Alagoez, F., Valdez, A.C., Wilkowska, W., Ziefle, M., Dorner, S., Holzinger, A.: From cloud computing to mobile Internet, from user focus to culture and hedonism: The crucible of mobile health care and Wellness applications. In: 5th International Conference on Pervasive Computing and Applications (ICPCA), pp. 38–45 (2010)
4. Schnedlitz, J.: Total Customizing - Vom Überleben durch Anpassung: The Eagleheaded Chicken Project. Berlin Press, Berlin (2012)
5. Holzinger, A.: On Knowledge Discovery and interactive intelligent visualization of biomedical data. In: Challenges in Human–Computer Interaction & Biomedical Informatics DATA - International Conference on Data Technologies and Applications (in print, 2012)
6. York, J., Pendharkar, P.C.: Human-computer interaction issues for mobile computing in a variable work context. International Journal of Human-Computer Studies 60(5-6), 771–797 (2004)
7. Ziefle, M.: Effects of display resolution on visual performance. Human Factors 40(4), 554–568 (1998)
8. Holzinger, A., Kosec, P., Schwantzer, G., Debevc, M., Hofmann-Wellenhof, R., Frühauf, J.: Design and Development of a Mobile Computer Application to Reengineer Workflows in the Hospital and the Methodology to evaluate its Effectiveness. Journal of Biomedical Informatics 44(6), 968–977 (2011)
9. Zigurs, I., Buckland, B.K.: A theory of task/technology fit and group support systems effectiveness. Mis Quarterly 22(3), 313–334 (1998)
10. Gebauer, J., Ya, T.: Applying the theory of task-technology fit to mobile technology: the role of user mobility. International Journal of Mobile Communications 6(3), 321–344 (2008)
11. Kraft, P., Rise, J., Sutton, S., Roysamb, E.: Perceived difficulty in the theory of planned behaviour: Perceived behavioural control or affective attitude? British Journal of Social Psychology 44, 479–496 (2005)
12. Yuan, Y.F., Archer, N., Connelly, C.E., Zheng, W.P.: Identifying the ideal fit between mobile work and mobile work support. Information & Management 47(3), 125–137 (2010)
13. Canalys: Smart phones overtake client PCs in 2011, `http://www.canalys.com/newsroom/smart-phones-overtake-client-pcs-2011` (last access: June 01, 2012)

14. Chae, M., Kim, J.: Do size and structure matter to mobile users? An empirical study of the effects of screen size, information structure, and task complexity on user activities with standard web phones. Behaviour & Information Technology 23(3), 165–181 (2004)
15. Farooq Ali, M., Pérez-Quinones, M.A., Abrams, M.: Building Multi-Platform User Interfaces with UIML. In: Seffah, A., Javahery, H. (eds.) Multiple User Interfaces: Cross-Platform Applications and Context-Aware Interfaces, pp. 93–118. Wiley (2001)
16. Stocq, J., Vanderdonckt, J.: A domain model-driven approach for producing user interfaces to multi-platform information systems. In: Advanced Visual Interfaces, AVI 2004, pp. 395–398. ACM (2004)
17. Mori, G., Paterno, F., Santoro, C.: Design and development of multidevice user interfaces through multiple logical descriptions. IEEE Transactions on Software Engineering 30(8), 507–520 (2004)
18. Choi, Y., Yang, J.S., Jeong, J.: Application framework for multi platform mobile application software development. In: 11th International Conference on Advanced Communication Technology, ICACT 2009, pp. 208–213. IEEE (2009)
19. Android: Android 4.0 Compatibility Definition, `http://source.android.com/compatibility/4.0/android-4.0-cdd.pdf` (last access: March 03, 2012)
20. Android-developers: Screen Sizes and Densities, `http://developer.android.com/resources/dashboard/screens.html` (last access: March 15, 2012)
21. JQuerymobile, `http://jquerymobile.com` (last access: March 15, 2012)
22. Phonegap, `http://phonegap.com` (last access: June 01, 2012)
23. Holzinger, A., Searle, G., Nischelwitzer, A.K.: On Some Aspects of Improving Mobile Applications for the Elderly. In: Stephanidis, C. (ed.) Universal Access in HCI, Part I, HCII 2007. LNCS, vol. 4554, pp. 923–932. Springer, Heidelberg (2007)
24. Holzinger, A., Trauner, J., Biffl, S.: Work Lists for the Transport of Patients: A case for mobile Applications. In: International Conference on e-Business (ICE-B 2008), pp. 454–459 (2008)
25. Holzinger, A., Mayr, S., Slany, W., Debevc, M.: The influence of AJAX on Web Usability. In: ICE-B 2010 - ICETE The International Joint Conference on e-Business and Telecommunications, pp. 124–127. INSTIC IEEE (2010)
26. Mesbah, A., van Deursen, A.: Invariant-based automatic testing of AJAX user interfaces. In: Proceedings of the 31st International Conference on Software Engineering, pp. 210–220 (2009)
27. Beck, K.: Test Driven Development. By Example. Addison-Wesley Longman, Amsterdam (2002)
28. Holzinger, A., Struggl, K.-H., Debevc, M.: Applying Model-View-Controller (MVC) in Design and Development of Information Systems: An example of smart assistive script breakdown in an e-Business Application. In: ICE-B 2010 - ICETE The International Joint Conference on e-Business and Telecommunications, pp. 63–68. INSTIC IEEE (2010)
29. Holzinger, A.: Usability engineering methods for software developers. Communications of the ACM 48(1), 71–74 (2005)
30. Holzinger, A.: Usability Engineering und Prototyping, Beispiel Mobile Computing. OCG Journal (Forschung und Innovation) 29(2), 4–6 (2004)