Enterprise Mobile Application Development

Strategies & Considerations for Building Mobile Apps

3/19/2012
A Blueranger Consulting Research Note
Author: David Bialer
dbialer@blueranger.com
415 425-9800

Q: What methods should enterprise IT clients employ for building mobile apps in the presence of multiple platforms?

By David Bialer prepared for Blueranger Consulting

Contents

Key Findings

- Broad multiplatform mobile device support is best achieved by using development tools and libraries that abstract platform-specific nuances, enable enterprises to create reusable cross-platform code, and reduce the breadth and depth of programming skills required so that a developer can create application code that avoids lock-in and reduces total costs of maintenance over time.
- Many enterprise applications, while complex in nature, do not require complex interfaces and can be written as web applications that can leverage AJAX-based interfaces to data, media resources, and business logic located at the enterprise data center or hosted in the cloud.
- Enterprises should evaluate their multiplatform mobile application project scope, desired range of device support, functional and stylistic application characteristics, and organizational requirements to devise an overall end-to-end strategy that separates a backend common technology implementation from the front-end technology used to create the application on the device.
- There are strengths and weaknesses of using different types of approaches to enterprise mobile development that balance tradeoffs between performance, variety of platforms supported, platform specific user experiences, code reusability, programming depth and breadth skills, application maintainability, and overall costs. A multiplatform application approach can be selected that best matches the enterprise requirements around these areas.
- While there is a lot of talk about HTML5/CSS3 application development to achieve multiplatform mobile device support, there remain incompatibilities between platform implementations, and using a pure HTML5 approach will limit the application to more-recent devices and future devices. There are development tools and programming techniques that can shield your organization from incompatibilities and create a forward-looking multiplatform strategy.

Recommendations

- No single multiplatform strategy is the right approach for all organizations or necessarily the best approach that can address a wide range of needs within or between enterprise business groups, but selecting an end-to-end solution, using open standards-based web technologies or a self-contained end-to-end proprietary system, can help most organizations design, develop, deploy and maintain mobile applications cost effectively and plan for future uncertainties.
- Organizations creating applications that are graphics intensive should consider using native platform programming tools or a cross-platform tool that has a common set of cross-platform interface APIs and generate native executable applications.
- Web-based applications that use HTML, JavaScript and CSS as their primary approach are best for newer iOS 4+ (iPad, iPhone), Android 4.0+ (Ice Cream Sandwich) Blackberry 6+, and Windows 7.5+ devices that have more, yet incomplete and inconsistent support for HTML5. However, using abstraction libraries, like the free jQuery Mobile or Sencha Touch, not only enable reusable code, but can also shield the organization from these incompatibilities, but at the price of limited device support and/or compromised user experiences.
- Self-contained proprietary, end-to-end systems (used in mobile content adaption and mobile portals) deliver high value when applications are not complex (largely displaying and collecting information and media), when legacy device support is needed, or when feature phone support is a requirement. This can be the case for many enterprise applications.

- If legacy or feature phone support is not required (which is a common expectation for mobile apps), and high performance is not required, Web-based programming methods are the best forward-forward looking multiplatform development approach for the vast majority of enterprise applications, as they will give you forward-compatible code, require less programmer skills, create better maintainability, and reduce overall costs over time. Using a hybrid wrapping approach today to web code enables code to deployed across a wide variety of devices, prevents vendor lock-in, and creates a maintainable applications that doesn't depend on specialized and scarce skills.

Analysis

What You Need to Know

The field of multiplatform mobile application development is rapidly evolving as more than 30 companies compete in this space to deliver tools, programming libraries and end-to-end self-contained solutions. Mobile application development is clearly evolving in the direction of creating web-applications that use HTML5, CSS3, JavaScript and other related modern Web standards, but ubiquitous and full support is not yet ready nor is it available on many devices already in use. The latest releases of Android and iOS-based devices are huge steps in this direction, but in the meantime there are intermediate and forward-looking methods that enterprises can use today to address a broader range of devices today and in the near future.

The vast majority of most enterprise application development projects should not need to worry about the nuances of these standards or compatibility between devices. Nor should they require different, often hard-to-find development skills for each platform (i.e. iOS developer, Android developer, Blackberry developer). Enterprises can rely on current, well-established libraries and frameworks that provide an abstraction layer to shield against device and browser incompatibilities. More important to the long-term success of a project is to carefully consider user and project requirements, application functionality, devices that the users will have, and organizational requirements in order develop a mobile platform architecture that recognizes that front-end technologies are mutable, replaceable and not mutually exclusive.

Background

Since the Apple iPhone was first introduced in 2007, the mobile device landscape has undergone a sea change in application platforms. This change has empowered the end-user to customize his/her mobile experience by allowing the end user to choose among many different operating platforms, form factors, features, and applications ("apps"). In the past, enterprises have been able to specify and standardize its mobile workforce on platforms of its choosing since email was the primary use case. Mobile applications for customers and partners weren't a concern. The landscape has evolved since 2007, and users not only desire to choose and use their own device for work (The "Bring Your Own Device" phenomenon), but enterprises are mobilizing their businesses as well. The major platforms used in smartphones, tablets, and feature phones are complex, competing, and incompatible.

Many enterprise departments, like product marketing groups, 'do their own thing' by developing their own mobile applications for their own stakeholders, sometimes very successfully, but this may decentralize corporate data and create challenges to managing costs and security. While these departments have business reasons to move forward with their own apps, they can cause risk to the security and the reputation of an organization, ongoing support and maintenance costs, and loss of

collective knowledge about the customer. These issues present serious challenges that have caused many enterprises to rethink and proactively plan their overall mobile strategy that addresses some common concerns:

- Can IT offer centralized mobile application development support across many different existing and yet-to-be-known platforms that each may support a variety of form factors, features, processor architectures, performance standards, and languages?
- Can the enterprise control and secure their corporate data from a central, secure location AND add value to departmental needs for mobilizing their products and services to their stakeholders?
- Can the enterprise innovate fast enough in supporting new usage cases, new devices and new platforms, and not become a bottleneck to the organization and cause departments to "work around" centralized efforts?
- Can organizations keep up with demand for mobile applications, skill sets needed to implement on each platform, and costs for supporting multiple platforms?

Why is this Important? The Matrix of Pain

Android-based mobile devices, iPhones and iPads have clearly emerged as important enterprise platforms as users have enthusiastically embraced these devices. Just iPhone and Android smartphones, taken together, are challenging enough to support. But there are other platforms in use, like the Blackberry and Windows Phone, and while their future demand may be unclear today, they will very likely play an important role in the mobile phone landscape. New or revamped platforms such as Blackbery 10, Windows Phone 7.5 (Mango), Windows 8, the questionable Tizen (Samsung/Intel), the rumored "Meltemi" platform (Nokia's feature phone replacement for the S40) may be just around the corner. Massively deployed feature phone platforms like Nokia's S40, fading platforms such as Symbian, and region or OEM-centric platforms like Samsung's bada(now merging into Tizen) or MediaTeks MAUI, could add even more platforms and form-factors that an enterprise will need to support.

The large variety of form factors (smartphones, tablets), input methods (keyboards, multi-touch, voice) and evolving device features (high res cameras, high-res displays, 3-D cameras, sensors) create thousands of device permutations that make it challenging to keep up with the existing devices, much less navigate the uncertainty of the future device possibilities. The rapid industry innovation could put at risk investments made in a particular platform, or make it prohibitively expensive to support and maintain applications across all the platform needs over the lifetime of an application.

Supporting each platform and device using native development tools (see Table 1) is a challenge in finding, training and retaining talented developers, and it is a challenge, as well, for an organization to master the design, development, test and maintenance of applications for all of the possible permutations, now and in the future..

Platform	Officially Supported Native Programming	IDE
iOS (iPhone, iPad)	Objective C	xCode
Android (Phone, Tablet)	Java (Dalvik)	Eclipse (Android SDK)
Blackberry Smartphones, Blackberry PlayBook, Blackberry BBX	Blackberry Java, J2ME C, C++ HTML5/CSS/JavaScript Adobe AIR, Adobe Flex, Adobe Flash Java (Dalvik) (Using Blackberry Runtime for Android)	Eclipse (Java SDK),Netbeans Eclipse (Native SDK), Netbeans Blackberry WebWorks SDK, Web Development Tools with Ripple Emulator Adobe Flash Builder Eclipse with BB for Plug-in for Android ADT
Windows Phone 7, Windows 8 (future)	Visual C# Visual Basic Silverlight XNA Framework	Microsoft Visual Studio with C#, Visual Baisc, Silverlight (XAML), and XNA Framework.
Symbian	C, C++ (with Qt), QML HTML5/CSS/JavaScript with WRT	Eclipse or Qt Creator Nokia Web Tools
Nokia (S40) feature phone	Java, J2ME Web Apps Flash Lite	Eclipse, Netbeans Nokia Web Tools Adobe Flash Builder
bada (Samsung), Tizen	C++, HTML5/CSS, JavaScript	bada SDK (with its IDE) Tizen SDK (when available)
Others: Brew WebOS (Enyo)	Brew (Qualcomm) Language Qt Webkit C, C++, HTML5	Brew MP SDK C++, QML Visual Studio, Eclipse, xCode

Table 1 – Developer Skills for Native Programming

All of these permutations together form the "The Matrix of Pain", a complex set of languages, environments, and platforms that requires different skill sets and different codes bases that just get larger as more platforms, form-factors, and application updates are releases. The Matrix of Pain = $[platforms * languages * IDEs * formfactors * applications * application\ revisions]$. Probably a tree really, but Matrix sounds better.

Thankfully, there are options that can relieve this pain and that, not only enable you to meet these new challenges, but to get ahead of the trends and provide a real service and forward thinking guidance to your departments desiring to mobilize their business.

Mobile Application Development Strategies in a Multiplatform World

The "Matrix of Pain" previously described has created opportunities for well over 30 companies and community-based solutions (i.e. open source) that offer a wide variety of approaches to solving these problems for both consumer and enterprise mobile applications. Each approach has its strengths and weaknesses, but there is most likely a close match to your needs.

The reason why there are so many different types of approaches is that there just isn't a single solution that would fit the needs of all companies or all needs within a company. But most likely there is a good choice for you and we will explain the considerations, approaches, strengths, weaknesses, and tradeoffs. There are several general classifications in approaches reviewed here that should be evaluated in context to your needs.

Solution Considerations

Before embarking on a multiplatform mobile strategy, it is a good idea to scope the problem you are trying to solve as this will have an important bearing on a chosen method. Important considerations you should look at before picking a solution:

Customer and project considerations

Consideration	Factor	Explanation
Scope of Project	How narrow or broad of a solution do you need?	A one-off or single application supporting a limited set of devices, or a company-wide initiative to service the vast majority of use cases in all regions and all potential devices? It is important to consider expected number of applications, life cycle (or longevity) of the application.
Audience	Who will be the users of the end-user application? Where will they be and what devices are prevalent? Are the users an internal audience or customers?	Internal users may be departmental and have a specific or narrow focus. The application may already exist and be deployed to desktops or accessed via browsers within the organization. How 'fancy' will the UI be? Also what regions of the world will it go into?
Developer Skills	Who will be the developers?	Some departments may have technical programming skill sets, while others have less technical skills and may desire a simpler drag-and-drop design tool requiring less skills, but often limiting functionality..

Device Considerations

Consideration	Factor	Explanation
Device Variety	Are there are broad range of devices that need support, or a few in particular?	There are several tools that support 2 or 3 platforms (typically iPhone and Android, and sometimes Blackberry, Windows Phone or Symbian), and these may be all you need.
Device range	Do you need to support smartphones and feature phones?	Smartphones are more 'platforms' and have a richer set of features available to the developer. Yet there are still many feature phones in use, but they may have limited functionality, smaller screens, and no touch interface.
Form factors	Does the application need to support smartphones, tablets, both?	Tablets generally have larger screens and may require scaling of images and videos. Some platforms can automatically detect and size media and have 'smarter' layout engines.
Geographies	Are there special geographic considerations?	Some areas like Korea and China have popular platforms not found in the North American or European markets (Samsung bada and Mediatek MAUI runtime environment in China, and variations of Android).

Application Considerations

Consideration	Factor	Explanation
Smartphone Features Usage	Does the application need to support security policy restricted features of the device or need graphics acceleration for complex animated image rendering?	Certain features, like media capture, need permission from the end user to use upon application installation, and may not be available to pure web applications. These include use of dialing, text messaging, camera usage, accelerometer usage, local storage, reading personal information like contacts, and others that vary by platform. Certain applications need to render using direct to hardware rendering technologies (like OpenGL ES).
User Experience	How complex or fancy is the user experience.	Certain user experiences, pizzazz like complex screen transformation for instance, may not be available across devices, but they may not be necessary.
Look and Feel	Should the application look and behave exactly like other applications on the platform (each platform has its own style), or should they look and behave similarly across platforms?	In general, the closer an application is to the 'native application experience', and the more platforms that it needs to support, the more limited its ability to be available across platforms. Sometimes applications can be made to 'look' native, but there may be some small differences.

Organizational Considerations

Consideration	Factor	Explanation
Compliance	Does the application need to comply with regulatory issues like application and data security, user authentication, and auditing.	Some application methods support these issues directly by keeping data within the corporate environment (in the backend) and have authentication built in.
Costs	What is the budget for development, maintenance of the applications and usage?	There may be lesser costs for upfront development, but ongoing costs for maintaining the application and/or hosting of the application.
Risk Management	How do you manage risk that a chosen methodology will have problems fixed? Will there be vendor lock-in?	Larger companies or more proven methods (i.e. using the native programming) may offer less risks to possibly getting dead-ended that smaller, innovative companies. Or open source technologies may (or may not) alleviate the risk of getting locked into proprietary solutions. There is also risk of consolidation happening in this space that affects risk (i.e. IBM acquiring Worklight, Motorola Solutions acquiring RhoMobile, Adobe acquiring Nitobi).
Licensing Policies	Will you organization use community supplied software available under open source licenses with no indemnification?	Some proprietary systems are available under commercial licenses that provide more warranties and indemnification.

A Best Practice Approach – Separate the Front-end from Back-end Development.

Though enterprise applications may be complex, their performance and user interaction requirements are often modest, and therefore may not require high performance user interfaces, business logic on the mobile device, or access to restricted device features (such as media capture). They are often connected applications that send and receive enterprise data and media from a remote server, and may have the presentation layer (the UI on the device) distinct and separate from the business logic, data, or media assets which may be based at a remote location on a server and accessed via an AJAX-based interface.

Much of an application's computing requirements and business logic can be handled on a backend, an area that is not as volatile and not subject to same refresh cycles of the consumer devices. As most new mobile devices have 3G, 4G or WiFi built in, fast and reliable connections have become more ubiquitous so many mobiles apps are, in fact, connected applications. This is the premise for modern smartphone platforms, where a limited amount of information may be stored on the device, but much of the business logic, data, and media assets are stored and managed remotely and securely.

The business logic components can have longer life cycles than the device and usually have common logic than can be shared across mobile platforms since this is typically device independent. Architecting a mobile application to split across a front-end on the device, and backend will decouple the business systems from the presentation on the device, which is where most, if not all, of the multiplatform requirements can be isolated

All the backend functionality can be accomplished using standard, reusable, scalable, and relatively future-proof technologies. Focus on an end-to-end solution that uses common technologies in the backend based on an AJAX framework or other vendor-supplied system. Focus your cross-platform efforts on the front end.

Choosing a Front-End Mobile Application Technology

While each platform has a 'native' programming language (see table 1) and a set (or multiple sets) of platform-vendor supplied tools, it is often challenging for an organization master and support all of these platforms, especially if there are more than two platforms or two applications being supported. Each platform has different methodologies and toolkits used to create, build and maintain an application.

Emerging cross-platform standards like HTML5 make it easier for developers to create innovative and appealing applications across multiple platforms and form-factors, but they are not yet fully mature and feature complete on all devices. They are currently most mature on iOS and the newest Android based devices (based on the Ice Cream Sandwich release – Android 4.0). Developer tools and platform abstraction programming libraries (like JavaScript libraries) that support these standards are abundant, and these tools shield the developer from having to understand platform differences and nuances, but may limit the number of platforms that can be addressed as not all features are available on all platforms, so they may either support a few platforms or support many using the least common denominator functionality that can compromise the user experience especially on higher-end and/or new devices. Other tools and libraries are available that use proprietary end-to-end technology, that support many devices, but may limit the features of the application.

Approaches

There is a wide spectrum of approaches to developing cross platform mobile applications. Some approaches use a combination of techniques. We will review several general methodologies each with a number of variations, though vendors in each category may have their own innovations that go beyond these descriptions. A comprehensive list and review of all vendors is beyond the scope of this research note.

Native Application Development

Native applications are applications that run as a binary or byte code executable on a device. Native applications have the advantage of giving the highest performance, allowing applications to access restricted features on the device (subject to security policies), and allowing the developer create applications on a platform with a native-look-and feel that is distinctive for each platform.

Native Application Method – Using Native Tools

Native platform tools (SDKs) are available for download (and described in table 1) at little or no cost via Android.com, Apple, RIM, Microsoft, Nokia, and others. Each one though requires somewhat of a different skill set and each platform supported requires its own development, test, release, and maintenance track. As there are no abstraction layers or interpreters, each platform may use different languages, development environments, application lifecycle methods, and library APIs, so other than the business logic, there is little reusability of code between platforms.

Native Application Method – Using Cross-platform Code Generators

There are tools and libraries that allow developers to code to a standard set of APIs and generate platform-specific high-performance executable versions (i.e. MoSync Pyramid, Appcelerator Titanium, Unity, Marmalade) thus enabling reusable code across different platforms. These tools may have limited platform support, or a limited set of features that they support really well, such as 3D rendering. They typically support at least iOS and Android devices. Some tools, like Unity3D and Marmalade, are used in game development across the popular smartphone platforms and are powerful for creating standalone, graphics-intensive applications, but are not as robust for or oriented towards creating web applications. They may use proprietary layout tools, proprietary scripting languages, C, or C++ (or really anything) as the development methodology, so a high degree of skill is needed to use these tools.

Mobile Web Methods

A mobile web approach uses standard web technologies, generally HTML, JavaScript, CSS and/or XML-based scripting languages to render the user interface. The code may be actually located on the device itself and seen as a local application, or be in the cloud, or any combination.

Mobile Web Method - Mobile Websites (local and remote)

Desktop, mobile web, and consumer electronics browsers are converging around a few browser layout engines: Webkit (Safari, Chrome), Mozilla Gecko (Firefox), Microsoft Trident (IE), and these engines are all in the process of implementing the HTML5 specification. HTML5, while not finalized (nor might it ever be really finished as it will continue to evolve), has enough commonality and active deployed browsers that support it, to make it a viable option for just addressing newer smartphones and tablets. While some developers may prefer to code directly in HTML5 and JavaScript, it can be tricky navigate the plethora of browser versions supporting different HTML5 functionality, and UI controls and touch panel support would need to be customized to match the native controls on the device (like scrollbars).

Mobile website scripting code may be local (on the device) or remote created with an AJAX framework, or any combination. JavaScript frameworks such as jQuery Mobile, Dojo mobile or other mobile-oriented JavaScript framework takes a lot of work out of implementing all the UI controls. These libraries provide platform-specific styling that approximates the native look, UI control libraries, touch event support, and other complex functionality. Application code is reusable across devices and form factors. These frameworks also shield the developer from concerning him/herself about which particular features of HTML5 are implemented in a browser engine. Commercial companies, such as Sencha (Sencha Touch) provide comprehensive and commercially supported functionality.

The advantage to building a mobile website is that web developer skills can be applied across many different platforms, and a smaller development team is needed to build and maintain the application. Yet there are some limitations:

- Functionality limitations – Applications may not be able to access the camera, contact list, or other device features due to the security policy on the device.
- UI limitations – may not look appropriate or like other applications on the device despite the fact that the libraries try to style for device. Soft buttons, menus, and other controls are just different on different platforms.
- Device limitations – JavaScript device and platform support may be limited to just a few platforms with advanced browser features or that use Webkit as the browser engine. You may not have control over which devices are used for accessing the mobile website.
- Inefficiency - Applications may have higher overhead than native applications (startup time, power consumption).
- Restricted distribution - Apple will not allow pure web applications in their iTunes store. So on Apple devices, mobile websites must be accessed via the built-in Safari browser.

Mobile Web Method - Mobile Portals

Mobile portals are websites specifically designed for mobile devices. They are typically end-to-end solutions and as an end-to-end solution, content is served and optimized for specific devices – so that the server is aware of the capabilities of the device and dynamically serves pages and scripts with appropriately laid-out content. They may use a proprietary markup languages and design tools. While supporting a large variety of devices and lower end devices, mobile portals may be limited in functionality to components supported by the vendor of the system. They offer reusable code and support a large variety of platforms, though you will most likely pay fees for tools and server licensing (or cloud hosting).

Mobile Web Method - Mobile Content Adaptation

Content adapters take content from existing websites and convert it, either dynamically or statically (or a combination), for the mobile web for a specific device. Like portals, content adapters keep a database of devices capabilities, but unlike portals they can use your existing website rather than creating a special mobile website. Content adaption may have limited features but a broad range of device support, and existing website code can be reusable – so it may take very little expertise to develop and low costs to maintain.

Hybrid Methods

Hybrid approaches combine native functionality with browser layout engine capabilities and allow developers to create native applications using web tools and libraries. There are several different hybrid approaches.

Hybrid Method - Application Wrapping

Applications are developed using web-based technology, including JavaScript Libraries such as jQuery Mobile (which seems to be the most prevalent), and 'wrapped' in a container that makes it a native application. It uses the layout engine to render the content and plays on a trick that native applications can call "web views".

PhoneGap (aka Apache Foundation's Callback) is one such prevalent wrapping technology used in a variety of off-the-shelf web development tools and SDKs. While layout capabilities are still limited by the browser engine support for HTML, developers can use standard JavaScript libraries (and create their own libraries) that allow the wrapped application use native such as the camera (subject to the security policies of the device). It also implements standards developed by the W3C for cross platform functionality (these are expected to eventually migrate to the browser engines).

There are a number of open source and third party development tools, JavaScript libraries, and plug-ins (JavaScript bridges to native functionality) that can be used with wrapping technology including jQuery Mobile and Sencha Touch libraries. A few vendors (Sencha, Exadel, Applcaton Craft, appMobi, Salesforce.com) support both mobile web application development and native application generation using wrapping libraries and tools.

Hybrid Method - Web Code Transcoders

Web code transcoders, tools that take web code and generate native code, are another way of cross-platform native code generation, are not technically web applications, they just allow you to code with web standards and perhaps reuse this code or parts of the code. The major advantage is that they leverage web programming skills to create applications but run with the performance advantage of native code. Appcelerator Titatium and Mosync Wormhole are commercially supported development systems that enable transcoding. They take somewhat more skill to use and have some limitations, either in the number of platforms supported or native features available through, but are useful for creating fast applications doing complex UI rendering.

Custom Webkit Engines

Another method of using web programming to create applications is to use customized versions of Webkit that include capabilities to access native functionality and act like a native application. This is a new technology, largely viewed as an intermediate step until standards are set and browsers ship in devices that implement proposed W3C standards that address using native device features securely. These standards around accessing secure device functionality from a browser are not finalized or implemented consistently. appMobi has one of the first technology custom Webkit engines called appMobi Mobius that only currently supports Android devices, so this technology is still a bit early. Other mobile web techniques (like wrapping) would need to be used on iOS devices.

Summary of Available Methods

The table summarizes some of the strengths and weakness of each of the approaches discussed.

Method	Project Scope	Device Requirements	Application Support	Organizational	Examples
Native Platform Tools	Good for a project narrow in scope on 1 or 2 platforms as that there is little reusability of code across different platforms. Highly skilled developers are needed for each platform that have skills in Java, C, C++, Objective C, C#. It could require many resources for multiple platform support.	Supports a single platform across range of high-end to low-end devices on any form-factor.	Rich features can be implemented, rich user experience with native look-and-feel. There is no feature compromise.	No built-in features for helping with compliance. High cost for a single application implementation as no time-to-market advantage. Expensive to maintain as application may need to be updated for each platform with new platform releases.	Apple - Objective C, Android Java

Method	Project Scope	Device Requirements	Application Support	Organizational	Examples
Cross-platform to native code generators	Good for a narrow projects for internally or externally distributed apps. Requires skill set investment in platform understandings and x-platform libaries. Medium amount of resources.	Can support multiple platforms across a range of high-end to low-end devices in any form factor.	Compromises on rich features as dependent on library to implement those features, or have to write native code for each platform. Can support a rich user experience and native look-and-feel.	No built-in features for helping with compliance. Medium cost for a single application implementation as some time-to-market advantage. Medium-High costs to maintain as application may need to be updated for each platform with new platform releases.	Mosync Pyramid.
Mobile Website	Good for single project for internal or external audiences. Can pin to screen as a bookmark to a website. Low development skills needed.	Can work on any device, though may not deplay correctly or the same for each device. If using HTML5, need to be careful on which features are used. May limit range of devices supported (as others may get an error), but supports wide geographic phone variations.	Limited features as no access to secured device feature (i.e. can't access contact list). User experience not tailored to device, looks like a mobile website, but this may be acceptable.	Supports browser encryption and security features (i.e. SSL), but cannot access restricted device features. Inexpensive to maintain and low risk.	HTML5/CSS3 with jQuery Mobile, Dojo Mobile, xUI. Sencha.
Mobile Portal	Good for multiple projects for internal and external audiences. Low developer skills and medium learning curve for proprietary language or tools. Low amount of resources.	Support for a wide variety of devices. Some scale and essentially proxy to translate newer HTML5 code to be used on lower-functioning browsers.	Medium feature set. Lots of 'widgets' that are pre-built and implemented across platforms. Can approximate look & feel of device. May have access to secure features on device if native client is generated.	Supports security policies of device, so typically feature limited. May have analytics and security features on server. Low cost to maintain applications.	Netbiscuits, Kony Solutions, July Systems, fitml.com
Mobile Content Adaptation	Good for multiple projects for internal and external audiences. Low developer skills needed, small learning curve. Low amount of resources.	Supports a large variety of devices and customizes content to the device. Large range of devices from legacy feature phones to newer devices.	Medium to low feature set. Adapts content to devices, but may not support all features on the device.	Same as web browsers. Low cost to maintain applications. Typically ca not access or use restricted device features.	Volantis, Infogin

Method	Project Scope	Device Requirements	Application Support	Organizational	Examples
Hybrid Wrapping	Good for multiple projects for internal and external audiences. Low developer skills though must know how to use a JavaScript library or development tools. Some expertise needed in at least one platform. Low amount of resources.	Can work on any device, though may not deploy correctly or the same for each device. If using HTML5, need to be careful on which features are used. Custom plug-ins are platform specific and may not be available on all platforms. May limit range of devices supported (as others may get an error), but supports wide geographic phone variations.	Rich feature set with close to native look-and-feel. Device feature usage subject to security policy.	Supports policies of device. Some vendors have enhanced security and compliance options and libraries for authentication (i.e. OAuth2).	PhoneGap in Application Craft, Adobe Dreamweaver, Exadel Tiggzi, appMobi XDK, appGeyser.
Hybrid - Web to Native Transcoder	Good for multiple platforms, but investment needed in learning languages and platform specific tool sets. Medium developer skills needed, but few resources.	May support a limited set of platforms (Appcelerator - currently iOS, Android, BB).	May have rich features sets and rich user experiences with a native look and feel.	No special compliance but may have 'plugins' for authentication and uses device security policy. Medium expensive to maintain.	appCelerator Titatium, moSync Wormhole.
Hybrid - Custom webkit	Good for multiple projects and uses standard web-based technologies. Low-medium developer skills needed, few resources.	May support a very limited set of platforms as it depend on a custom Webkit implementation for that platform.	May have rich features, but close to native look and feel. Device feature usage subject to security policy of device.	No special compliance but may have 'plug-ins' for authentication and uses device security policy. Medium expensive to maintain.	appMobi Mobius.

Tradeoffs

There is a balancing act around managing the priorities of your multiplatform mobile development requirements. Tradeoffs that may need to be made – usually between the breadth of the solution (in terms of devices, degree of programmer skill or expertise, and reusability of code) and performance, device-specific UI design, and maintainability. Though this isn't always the case and the tradeoffs narrow as you narrow the scope of your solution, here are some guidelines.

- **Limited number of platforms (1 or 2**) - If broad multiplatform support is not an issue (e.g. you are absolutely sure that you will only need iOS and Android support), then native programming is the way to go, that is, if you have the resources and skills. You can expect vendors like Apple and Google (for android) to enhance, maintain and update their tools so there is little risk of getting dead-ended by the platform provider or tool supplier.

- **Performance** – If your applications have complex graphic animations, fancy transformation or strict performance needs, then you may need to bite the bullet and use native programming tools, or a native code generator. You may have to write the some of the device-specific feature support yourself, or license from a third party. But you can use tools like that take web-based code and generate native code here as well.

- **Large number of platforms** – If you need to support a large number of platforms, three or more would be a good rule of thumb, then using a web programming technique or hybrid method would be the best approach. You may just develop a mobile website if you don't expect to need to use any device-specific features (like accessing contacts, media capture, accelerometers, local storage, SMS, dialing). It should be noted that Apple currently does not allow distribution of web code through its iTunes store and has strict guidelines around this (so on Apple devices, users would have to access it via the Safari browser and bookmark it).

  If you want your application look good on even low end devices, and not worry about testing across a lot of devices, then a mobile web portal method should be considered. With a limited amount of resources, a content adaptation method might be better as it will save you having to manually code a separate application or mobile website.

- **Best Choice** - Otherwise, it is my opinion that the best forward looking method, especially if you care more about future devices than supporting legacy devices and your applications could possibly use restricted features, use a hybrid method like wrapping as, if and when mobile browsers really support usage of device features securely, you will easily be able to adapt and move your code forward to the next generation of devices and it will prevent you from a vendor lock-in as your application can be based on completely open standards.