intelliware software development

Mobile

Going native: cheaper than you may think

A commonly held belief is that writing native mobile apps is expensive, especially when you need to support multiple platforms. A related belief is that writing apps using HTML5 or a Mobile Application Development Platform (MADP) tool entirely solves the cost issues associated with cross platform application support.

The hard truth is: getting an app to run effectively on multiple mobile OSs using any of these methods is more complex than you may realize. If your goal is to optimize the end-user experience, the cost difference between writing native apps tailored for each mobile OS experience and writing a generic app that runs properly in all mobile environments may be less than you expect.

The right time and place

First, let's differentiate between the two most common approaches to non-native development: HTML5 and Mobile Application Development Platforms.

HTML5 mobile development was originally used to bring largely non-transactional websites to mobile devices. However, the sophistication of mobile applications has evolved along with HTML5, and now mobile sites hook into back-end systems through SOAP or REST APIs to provide real-time data and transactional support. Usually, these are combined with a JavaScript framework like jQuery Mobile to make the front-end development easier and to provide developers with tools to deliver a rich user experience. Even as HTML5 device processing capabilities have evolved and improved, they still face two significant shortcomings. One, for a pure HTML5 app to work, it requires always-on connectivity. Offline processing is not possible without some tricks, and network latency can introduce application performance problems. Two, to take advantage of mobile online stores, you'll need to provide at least some kind of a minimal native "wrapper" application to establish a presence online. These two shortcomings make the pure HTML5 option decision a difficult one to make.

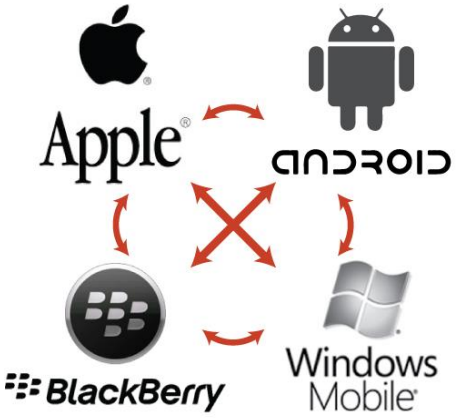
> "If you are truly platform agnostic, and will not be using any of the native design or interface elements, then there is no need to optimize for each OS."

Mobile Application Development Platforms represent a wide range of tools that generally encompass an editing suite, an IDE (integrated development environment, often based on Eclipse), wizards and data access protocols. These packaged tools promise speedy delivery and the capability to write the code once and deploy it 'everywhere' – in other words, any target mobile OS. There are about a dozen MADP tools that can be considered as major brands.

A perfect fit for MADP or HTML5 development are apps that seek to establish their own unique UI look and feel. Games and financial trading apps are good examples. If you are truly platform agnostic, and will not be using any of the native design or interface elements, then there is no need to optimize for each OS. Between the two approaches, the accepted wisdom is that if your app is complex and has extensive data needs then investing in a MADP tool is preferable to pure HTML5 development.

The reason for this is what most MADP tools deliver in the way of server integration. Products such as Kony and Worklight include a server installation where developers can easily add mobile connectors which allow the mobile client app to communicate with any back-end enterprise data a company may want to surface.

MADP products go beyond HTML5 in that they offer more than just a common language for cross-platform mobile development. MADP products include things like Mobile Back-end-as-a-Service (MBaaS) and Mobile Device Management (MDM) among other things, allowing companies to not only unify their development on a common set of tools, but also better manage their development after an app is written. They also enable better management of how an app integrates with existing enterprise data stores. There are definitely situations where this type of service is the way to go, and we will see more and more projects adopt this technology.



Business reasons may tie you to a MADP or HTML5 strategy. Perhaps your company is invested in these technologies, perhaps your development resources are limited or the app has to appear on the market on multiple platforms at the same time. Simultaneous release is achievable with other strategies, but it is easier to achieve with MADP or HTML5 and their minimal reliance on the native elements of each target OS. Be aware though, you will still need to spend a significant amount of time polishing and customizing your app for each OS.

While it's generally easier to manage one team instead of many, the reality is getting to market on multiple mobile platforms simultaneously will require multiple teams - one per platform, working in parallel. With this reality comes more complex project management and increased costs.

It's never a perfect world

Although there are benefits to using a multi-platform development tool strategy, there are situations where they are not as effective as advertised. The biggest challenge with MADP and HTML5 strategies is that they almost always require additional effort and tweaking to "get it right" across different mobile environments.

The biggest weakness of MADP and HTML5 is they cannot deliver the responsiveness and familiarity of the native user experience. Frameworks like JQuery Mobile or Sencha Touch do make it relatively quick and easy for developers to write native-like mobile apps. Unfortunately the JavaScript engine in the latest version of every major mobile operating system is lacking and these frameworks always fail to respond to user touch events as instantly as native apps. HTML5 apps feel sluggish compared to native apps and no amount of optimizing or architecture magic will fix this problem.

Until the performance of the JavaScript engines in mobile browsers and frameworks, like JQuery mobile improve, performance will continue to be an issue. Some MADPs avoid this problem by compiling code you write into native code for each platform, but this also usually means being limited in what features you can implement. For example, Kony is designed to provide a tab bar interface on iPhones for any "Menu-ed" app, which is great if you want a tab bar interface. However to push that menu into something that slides out from the left on an iPhone, the behaviour Kony provides by default for Android from the same code base now requires you to write special code to handle the menu differently on the iPhone so that it matches your design. This translates into a huge chunk of extra work – and that's just to handle one platform properly, negating the cost and complexity savings of going with a MADP in the first place.

"HTML5 apps feel sluggish compared to native apps and no amount of optimizing or architecture magic will fix this problem."

Hardware differences between manufacturers are starting to disappear, so the only real way manufacturers can differentiate among each other is through a "unique user experience." The announcement of iOS 7 showed us a new graphic design and more importantly, new APIs for developers to add in physics-based UI elements. BB10 came out with unique bezel gestures, while Android added accelerometer gesture detection. These are just some of many examples, and each requires you to adjust or optimize your code to use these features.

The problem grows even more complicated when you consider that most Android manufacturers "skin" Android with their own UI, adding their own unique features. Android on a Samsung phone looks and behaves a lot differently than Android on an HTC phone – neither of which bears any resemblance to the stock Android you'll find on a Google phone. You'll have to optimize your code for each flavour of Android, in addition to the cross-platform optimizations.

You could choose to have your app use only your own interactions with the device or opt for a product like Nanogest for iOS and Android that uses front facing cameras to capture command gestures in the air above the device. Still, the bottom line is customers choose their devices for a reason – cost, design or, most importantly, user experience. They are expecting to be able to use the lexicon of swipes, motions or gestures native to their chosen device, and not have to learn new ones. Above all, they expect your app to be just as fast and responsive as the rest of the "rich user experiences" that the native elements and OS provide them.

So here you are, stuck translating your apps to all the platforms, and often versions of those same platforms. What will make a significant difference to your deliverables is to focus your design and development decision process. Let us assume that the development costs will be managed and kept under control, regardless of whether you take a MADP, an HTML5, or a native approach. You should be free to choose your development platform based on what will allow you to deliver the best possible user experience. Ultimately that's where you want to start and where you should focus your decision making process.

If user experience is a top priority (and it should be) then native development should be the solution. Developing three or four separate versions of the same app may sound like madness to you, but there are several ways to manage the costs and complexities.

The natives are friendly

We have argued that the more complex your mobile app the more time you'll need to resolve cross-platform issues. Gauging that level of complexity brings us to considering some of the reasons for going native. There are strong indicators for developing natively, including factors such as whether your app has:

- A high degree of user input
- High volume of transactional data
- A need to cache data
- The need to access the device hardware

So the question arises, how do you create multiple native versions of an app in a way that is fiscally viable? We believe the answer is a port of sorts. We think of it as re-implementation – basically, you write one native app and then, keeping as much of your first effort as possible, you create the next and then the next.

In some ways this may sound like porting, but that is not the case. Porting is an *automated* process of taking a single code source and recompiling it for different platforms. It is a shortcut. For example, you can use BB10 and Tizen's wizards to automatically port your Android app to their platforms. You will still have to tweak it and fix incompatibilities

"You should be free to choose your development platform based on what will allow you to deliver the best possible user experience. Ultimately that's where you want to start and where you should focus your decision making process."

because there's no perfect or magical solution. It may be fast, but auto-ported apps will be limited in their ability to use all of the native features of their new platform.

We suggest you take a single application concept and re-implement it using different languages, thereby saving time and money through reuse. The primary thing to retain and pass on is the app's design and planning, as well as the developer's experience of the first build.

The next element in a successful re-implementation is executing a development strategy that creates as many common components as possible, especially in the back-end. When you begin the next version of the app, you know what the design is; you just need to implement it to take advantage of the native OS elements. In a sense you allow your design to subsume itself into the native design.

Simultaneous release was stressed when we discussed MADP because what we are proposing are several, separate development cycles. Writing native apps for multiple platforms can be done much more cost effectively than previously thought by using a strategy of writing the app for one platform first, and then re-implement the design to other platforms. With each iteration developers will gain velocity as each rewrite deepens their understanding of the overall problem space, making design and decision-making hurdles quick to resolve.

What is important is that you gain advantage with having the same team work on the successive iterations – their knowledge builds each time. If your timelines make this impossible and you have multiple teams or employees with platform-specific skills, you can simultaneously run second iterations of development for the other platforms. It is important that all the teams benefit from the knowledge gained by the first team. Perhaps the members of the original team could be divided among the new teams to pass on their knowledge. These individuals could also form a communication link between the teams so they don't work in isolation.

Getting started with a native strategy

Your upfront planning will make all the difference

The best approach for going native is to choose one platform to develop your app on, and then plan and design with all platforms in mind.

Knowing at the outset that you are planning multiple releases will allow you to make decisions that can save time overall.

For example, even if you plan to create native apps, you might choose to still use HTML5 to present a number of common static screens within your app, like the About page. These pages can easily be shared across all your mobile projects, saving you time and money with a little up-front planning.

Re-implementing: a cost-effective approach to cross-platform development

From a developer's point of view, one of the greatest advantages to native development is the freedom from worry about cross-platform compatibility. You are able to use hooks and features unique to a specific OS without having to create workarounds for other platforms. Similar features (e.g. share) that are implemented differently across platforms are far easier to develop. The code is cleaner and easier to maintain. Above all, you can get the most out of each platform and present the user with the best experience, all without wasting time "optimizing" code so your app is responsive. Native apps are almost always responsive without effort. The compiler optimizes the native UI widgets for you and the native SDKs force you into sensible design patterns for the mobile platform.

> "The code is cleaner and easier to maintain. Above all you can get the most out of each platform and present the user with the best experience..."

There is a drawback: you are now supporting multiple code bases and all the maintenance that this implies. That's a pretty big negative, but it does hide an *advantage*: handling updates. In a multi-platform solution, an update to one OS may bring new conflicts that have to be resolved on the other platforms, which should not have been affected by that update. A native solution should give you less cause for maintenance in the first place, and OS updates should be much easier to apply to your app. The only time you will need to modify all your codebases is when your app is updated. Ideally, you have designed your app such that most of the functionality requiring updates is handled by the back-end. As mentioned earlier, the goal of your design is to have one back-end handling multiple native front-ends.

If you plan to pursue a native development approach, you may wonder where to start. Is there an ideal platform to start with? The short answer: it depends. On many projects your choice of platform will be constrained by the client's needs or market timing. Most likely, the

answer will come from business rather than technological drivers – your existing customer base, your partners, and who is the current or emerging market leader. In an ideal world your technical capabilities would have equal weight. If you have an in-house team or a development partner, consider their strengths and expertise and ideally, your business needs and development expertise will align.

Native with a side of re-implementation: a case study

As a case in point, the following provides an example of the benefits of taking a native approach to a new app, then re-implementing across various platforms.

In one of our major mobile development projects we were hired to create a consumer focused, data rich iPad app. Several months after we delivered the project, the client wanted to release a new, native version of their app for the BlackBerry PlayBook launch. After that success, our client wanted to take their app into the BB10 market, so the third re-implementation created another native version of the app. This process of successive iterations became a development laboratory. The most important lesson we learned was with each re-development iteration, we saw the project time (and associated costs) shrink significantly.

> "The most important lesson we learned was with each re-development iteration, we saw the project time (and associated costs) shrink significantly."

Our analysis showed that we realized the greatest time and cost savings by having a focused planning and design strategy that we reused for each version of the app. Additionally, the team was able to work faster on the later iterations. Having gone through the process once, and having encountered and solved issues the first time, we did not have to re-think challenges nor reinvent the wheel. Above all, developing natively we did not have to struggle with contradictory code and OS requirements.

Let's look at the specifics of how costs can be lowered when you re-implement and evolve an app from one platform to another. Although three different "native" languages (iOS: Objective C, PlayBook: Adobe Air, and BB10: C++) were employed in these projects, our strategy allowed us to use and share the same data structures among the different implementations. Smart application layering made the work of building out support for new platforms relatively straightforward.

Initially, we expected each project to have similar costs since they had the same scope and complexity. In particular, we believed the BlackBerry 10 version would be costlier to develop given the added complexity of rethinking the UI for a smaller screen. To our surprise, the development times shrank with each version of the app we developed.

The iPad version took a total of approximately 270 developer days of effort. This included about 50 days on user experience (UX) design and prototyping, and another 50 days spent on development planning. Not surprisingly, as this was the first version of the app, fully one third of the development time was spent on planning and design to find the best approaches to every aspect of the project.

> "Above all, developing natively we did not have to struggle with contradictory code and OS requirements."

Despite the size and complexity of the two projects being roughly the same, the PlayBook version was completed in just 120 developer days of effort. The development time savings between these two projects can be attributed to having settled on an overall look and feel for the application, and a large degree of overlap between general tablet application conventions. However, some of the tablet use conventions are dissimilar. For instance, menu handling and swiping gestures can have very different meanings on an iPad versus a PlayBook. Using each device's native development platform allowed our team to be focused on building the right experience for that device. In a MADP or HTML5 environment much greater effort would be required to make each application version behave in a way that the user expects from their device of choice.

Another key to our ability to successfully re-implement the original app was in the initial iPad application design. Data structures, content management and other shared features were implemented outside the native application in a way that made them sharable between the different platforms. This cut our overall development time in half.

One year after the PlayBook version was published, we re-implemented the app for the launch of the BlackBerry 10 devices. It was critical the BB10 version had all the features of the latest PlayBook and iPad versions. This meant our phase 1 project scope was close to double in size compared to the original tablet projects, as we had continued throughout this time to add features to the iPad and PlayBook versions. In addition, we had to include integration with the client's campaign server to promote featured content without requiring an app update. We even added unique BB10 features such as the ability to share the app through BBM, and built a unique grid view/list view toggle for the main app screen to better present content on the smaller phone screen.

Despite nearly doubling the scope of the project, our team took even less time on the BB10 project than the PlayBook re-implementation, completing the initial version in just 90 developer days. That's a full two-thirds less time than it took us to create the original iPad release, and a third faster than the effort we put into the PlayBook version. Even more remarkable – we spent only 10 days on design and planning, even though we were not only moving the app to a new mobile OS, but also from a tablet to a phone.

In addition to the team's now extensive experience with the app, a contributing factor to this even greater reduction in the BB10 development time was BlackBerry's native SDK framework, Cascades. It is clear that BlackBerry sought to make writing fast, attractive native apps for BB10 as easy as possible. Find out more about our BB10 experiences here.

Conclusion

At the end of the day, if people are engaging with your brand through your app, then it is critical to avoid any compromises with the user experience. A bad experience has many perils – poor brand impression, lost customers, fewer transactions/sales, etc.

Your design, product and marketing teams, not your technology choices, should drive how your app looks and behaves. You don't have to go native to build a first class app, but depending on what you're trying to deliver, native often makes it easier.

If your customers demand a rich app experience, "going native" may be the right choice.

Contact us if you want to learn more about how Intelliware can help with your mobile solutions strategy or development.

intelliware
software development

About the Author

Marc Henderson

With over 15 years of software development experience, Marc has a proven track record building custom enterprise software solutions at global organizations. In addition to his enterprise software development experience Marc has a strong design background. He began his career at one of Canada's largest educational publishers where he designed educational materials and wrote interactive, scholastic CD-ROM and web-based games. Throughout his career Marc has been interested in leading-edge technologies with a particular interest in mobile. His recent work history includes many years in mobile application development and research that follow key evolutions in the mobile ecosystem.

About Intelliware Development Inc.

Intelliware is a custom software, mobile solutions and product development company headquartered in Toronto, Canada. Intelliware is a leader in Agile software development practices which ensure the delivery of timely, high quality solutions for clients. Intelliware is engaged as a technical partner by a wide range of national and global organizations in sectors that span Financial Services, Healthcare, ICT, Retail, Manufacturing, and Government.

Intelliware placed among the Top 5 Mobile Technologies Companies in the 2012 Branham300 report, the definitive listing of Canada's Information and Communication Technology (ICT) industry leaders, as ranked by revenues.

200 Adelaide Street West, Suite 100
Toronto, Ontario M5H 1W7, Canada
416.762.0032

www.intelliware.com