# Going native (or not): Five questions to ask mobile application developers

James White

Curtin Health Innovation Research Institute, Curtin University

## EDITORIAL

Please cite this paper as: White, J. Going native (or not): Five questions to ask mobile application developers. AMJ 2013, 6, 1, 7-14. http://doi.org/10.21767/AMJ.2013.1576

**Corresponding Author:**
James White
Centre of Excellence for Science, Seafood and Health, 9 Parker Place, WA 6102
james.a.white@postgrad.curtin.edu.au

### Introduction

I recently needed a new shovel. At the hardware store I was, as always, taken aback at the range of options available to me, and the range of prices I could pay. Hardware store regulars will know that this is true of everything from lawnmowers to screwdrivers – one can pay an incredibly small price, an incredibly large one, or anything in between. In case you are wondering, I chose the second-cheapest shovel, a policy that has served me fairly well over the years.

Shovels, lawnmowers and screwdrivers are easy. We know what we are looking at. We can judge the look and feel, and we know exactly what we will have when we get it home. We may be familiar with the brand, and we will almost certainly be familiar with the materials; we know, for instance, that steel is harder and longer-lasting than plastic, but that it is also heavier in the hand. We can read a list of features, seek help for the items we do not understand, and make an informed choice.

Most health professionals who have ever commissioned a piece of software will know that it is a very different prospect. The healthcare industry is increasingly aware of the opportunities and benefits of information technology. This means that people who have never given a great deal of thought to the development of the software they use, increasingly find themselves discussing large sums of money with people who are wont to enthuse greatly about this or that approach to development, using this or that technology, in language that *almost seems like English*.

A colleague of mine recently sought quotes for a mobile application, and received responses ranging from $30,000 to $150,000. What are we to do when faced with such wildly divergent figures? How can we make a choice and have confidence that we will get value for money, that the project will be completed on time and to our specifications, and that we will end up with a quality product that matches our expectations, those of our funders and, most importantly, those of the end users?

There are no easy answers to these questions. But as someone with a foot in both camps – I am both a public health researcher and a software developer – I have some advice. Specifically, five questions you should ask the next developer who is eager to tell you exactly how you should spend your hard-won project funding. Naturally there are a great many more than five questions that could, and should be asked, but I consider that the majority of them can be formulated and understood by most people, or at least, most people who have ever commissioned *anything*. You will want to know how experienced the contractor is, and perhaps talk to their previous clients. You will want to know that they have experience in developing applications or "apps" with some commonality with your project. You will ask questions relating to timeframes, extra costs, guarantees, intellectual property and so on. The questions I propose here are related directly to the field of mobile app development, and specifically to the underlying structures with which apps are built – their DNA, if you like. To most people "an app is an app", and although they may be able to judge the good from the bad, they may be less able to pinpoint the characteristics that make it one or the other. These questions may assist in doing so, and in helping to insure that, when complete, your app falls into the former category.

However I am going to make you, the reader, work a little before giving you the questions. I will begin by describing some essential characteristics of mobile applications, and some important considerations. In digesting this, you will more than likely formulate a list of questions for yourself;

you can test your own comprehension by comparing them to mine, which I will provide at the end.

**Mobile apps are not born equal**

Like so many things, there are several ways you could categorise apps. You could reasonably say that there are five basic kinds of app, or three, or 20. I will say here that there are two types, or rather two ends of a spectrum. At one end are *native* applications and at the other are those variously called *web apps*, *browser apps*, or *non-native apps*. Each has pros and cons, and it is essential to know which your developer is proposing.

Native apps are built with a specific family of devices in mind. Presently, one could build a native app for iOS devices (iPhone, iPad and iPod Touch), for Android devices (a plethora of smartphone and tablet devices made by various manufacturers, which run on an operating system developed and maintained by Google), for Windows-compatible devices (Microsoft's latest Windows operating system is compatible with some third-party smartphones and tablets, as well as Microsoft's own newly released Surface tablets), for Blackberry, or for one of a few smaller players.

Each of these operating systems requires that native apps be built using a particular coding language. For those taking notes, it is *Objective-C* for iOS, *Java* for Android, and typically *C++* or *C#* for Windows. They also provide a set of protocols for accessing the various interface objects, functions, utilities, aerials and sensors of modern mobile devices. These *application programming interfaces* (APIs) give developers access to extensive frameworks and tools that are written by the platform curators, specifically for that platform. The use of these APIs for both visual elements and under the hood functionality conveys the native "feel" of an app. In addition, APIs enable developers to build apps which can directly access device features such as cameras, GPS aerial, accelerometer (the sensor that detects the orientation of the device), microphones, and so on. Non-native apps may be able to access some of these features, such as the camera or user location, but they do so using non-optimal methods.

Native apps are distributed directly by the companies which manage the operating systems, such as Apple, Google and Microsoft, via applications stores on the device, or on desktop computers. Upgrades and bug fixes are also managed in this way – developers who wish to modify their app must do so via a submission to the relevant application store, and wait whatever time that store takes for approval. Some platforms take a curative approach to distribution, requiring apps to be checked for functionality, security and content before being approved for distribution (Apple has been famously stringent in this respect), while others take a more hands-off approach.

At the other end of the spectrum are non-native web apps, designed to work across many devices and operating systems. They use common languages accessible on all devices, including *HyperText Markup Language* (HTML) and *Javascript,* languages used for general web development. These apps are essentially websites that have been optimised for smaller screens, although optimisation is a challenge when the developer is trying to support literally hundreds of different devices, all with different screen sizes, resolutions, central processing units (CPUs) and graphics processing units (GPUs). Users receive a URL address, just as they would for a standard website, and navigate to it using the browser on their device. The operating system and device manufacturers have no control over content or functionality – developers may make changes at any time, with immediate effect.

In the middle of the spectrum are so-called *hybrid* apps, which take web-based functionality and wrap it in native containers. This results in a set of native applications, one for each targeted system, sharing web-driven content. These are distributed via the appropriate application stores and, while some core functionality may only be altered via a new submission, other content may be updated immediately. There are also emerging technologies that enable developers to write an app using a single language, then to translate that code into native code for various devices. Perhaps the fairest thing that can be said about this approach is that "results may vary". The tools are improving all the time, and there have been some very good apps built using this approach. However, there have also been many that were demonstrably inferior.

It is very important to be clear about which of these approaches a developer is proposing. It is particularly important when dealing with this last category of hybrid or cross-platform apps, as there is great potential for confusion and misplaced expectation. A developer could say that a hybrid app, built using large amounts of web-served content, using a cross-platform complier, is native – it uses *some* native APIs and is distributed via the appropriate application stores. They could also make the case that this is the best of both worlds, and in some cases they may be right. However, if they *are* right, it is because this approach is an effective solution to the particular requirements of the app project under

discussion. Not because it is the best solution *per se*. It is vital to understand the advantages and compromises inherent in each approach.

**Advantages of native development**

The core advantage of native applications is that they are built according to a set of specifications provided by the operating system manufacturer. These manufacturers provide vast libraries of code which can be used by developers, and this helps to ensure some level of consistency across apps. Buttons, indicators, item choosers and navigation structures may all work consistently from app to app, because they are using the same code base, developed by the stewards of the platform, and refined over time. By contrast, an interface object in a web app may have been designed and coded by anyone, and will vary greatly from app to app.

The consequences of this variability should not be underestimated. It is remarkable to consider the extent that mobile devices have penetrated our daily lives, in a relatively short space of time. Many people use such devices very regularly through the day, for all manner of tasks, and as a consequence the interface of the device itself becomes very familiar to users. This fact is truer for mobile devices than it has ever been for desktop computers. In short, users expect apps to behave in particular ways, and there is an immediate disconnect when they do not. For example, many native mobile apps use a standard navigation structure to move from one screen to another. The device animates smoothly between the views and, because the content is usually embedded in the app, it appears almost instantaneously. Furthermore, the device presents various standard controls for navigating backwards and forwards through content – users recognise these controls, and know what to expect when tapping them.

Many web apps try to mimic this design and functionality, but even the very best examples cannot achieve more than an approximation. For one thing, because the content is loaded from the web rather than from within the app, it will typically take more time for new screens to load – sometimes significantly more. The experience is much more like viewing a web page, than using a mobile app. There is a school of thought that developers have made a rod for their own back by attempting to imitate native design; by trying but falling short, they have effectively set up false expectations for the user.

The means of loading content leads to another advantage of native development – all things being equal, a native app will consume far less data than a non-native, web-based equivalent. That is not to say that native apps consume no data – most modern apps, no matter how they are built, will access the Internet for some purpose or another. The critical difference is that, in a web app, *everything seen on screen* has been downloaded on the fly. By contrast, a native app will include a great deal, and in some cases all, of the data it needs to function, at the time it is first downloaded from the distributor. Some distributors place an arbitrary limit on the size an application can be, if it is to be downloaded over a cellular connection; large apps can only be downloaded over a WI-FI connection. This prevents an app from consuming an excessively large amount of a user's cellular data allowance at the time it is first downloaded. There are no such safeguards with web apps, and this can impact on both performance and cost to the user. It should be noted, however, that *good* developers will attempt to design web apps with this in mind, and it is certainly possible to develop efficient, fast, data-economic web apps.

Another consequence of the contrasting use of data is that with most native apps it is possible to use some, most, or even all of the app's functionality *with no Internet connection at all.* The app's content and programming code is contained in the app when it is first downloaded. It may also be programmed to detect the presence or absence of a web connection, and modify itself accordingly. If parts of the app require a connection, but the user is currently offline, the app may hide or modify those functions, or present the user with a notification about the need to be online to use that part of the app. No such niceties exist for web apps; they simply will not work.

Finally, the use of native APIs enables developers to give users the option of accessing information and services on their device outside of the application they are using. This includes contact lists, calendars, photo and media libraries, and shared credentials (such as those for social media). Apps that enable users to add an event to their calendar, send something to a contact, or use media on their device, typically do so using native APIs.

**Facebook: A case study (and cautionary tale)**

Some of the most high-profile mobile applications on any platform are those developed by the social networking behemoth Facebook. This stands to reason, given its enormous user base, and the degree to which social media usage has been one of the biggest drivers of smartphone uptake.

Facebook initially chose a predominately web-based structure for its mobile applications. The company

distributed apps for various platforms that were essentially native containers full of web content, built using the HMTL5 web standard. HTML5 is the latest set of specifications for *HyperText Markup Language*, the primary language used in web development. These specifications make it more suited to mobile development, and allow new functionality designed to reduce dependence on outdated technologies. One of the main reasons for Facebook's decision was a desire to be flexible. Facebook is a developer-driven company with a strong preference for agility and quick iteration. In other words, the company likes to try new things often. Some software companies are more cautious, pilot testing and perfecting new features over long periods of time before releasing them to the public. Facebook prefers to quickly develop new features and new ways of presenting content and to trial innovations with some parts of their massive user base before pushing changes out to all users.

This approach is problematic with fully native apps, where changes must be reviewed and approved by the various application curators before going live, a process which can take some time. Furthermore, it is difficult or impossible to make changes for one group of users, but not others, making Facebook's approach to large-scale beta testing unfeasible; if the test process breaks something in the app, it breaks for all 300 million of its mobile users, rather than just a "few" million.

So Facebook took a web-first approach to developing its mobile apps. The resulting apps were almost universally derided as slow, buggy, inconsistent, and prone to frequent crashes. There was some disagreement over why this was – the apps were bad because web apps are generically inferior, or simply because they were poorly programmed. Going straight to the source, Mark Zuckerberg, CEO of Facebook, expressed clear thoughts on the issue. He made the following comments at a technology conference in September 2012:

> *"The biggest mistake we made as a company was betting too much on HTML5, because it's just not there yet. We had to start over and rewrite everything to be native. We burned two years. It may turn out it was one of the biggest if not the biggest strategic mistake [we made]…We believed that because it used the same technology as the desktop, we thought it could improve. But it wasn't good enough. We realised the only way we could get there was to go native."* [1]

So Facebook began from scratch, building new, platform-specific native applications. The iOS version was released in August 2012, and showed a vast improvement in speed (up to twice as fast), stability and device integration.[2] Of course, the actual content (users' posts, comments, photos, video and so on) is still delivered via the web. The difference was that the scaffolding containing this content was written natively, and the tools used for accessing things like the device camera and GPS aerial, were built using native APIs. At the time of writing, a native Android application remains under development. On the day the new, native iOS application was released Mick Johnson, Facebook's iOS product manager, said:

> *"A native Facebook iOS app has been arguably the most-wanted app on the planet. It doesn't look much different, but should satisfy the hundreds of millions of users begging for an experience that isn't cripplingly slow."* [2]

Consider that the phrase "cripplingly slow" referred to his company's own product that was, just the day before this statement, used by more than a hundred million users as a primary access point for Facebook's content. This is a telling statement indeed.

**All native, all the time?**

It may appear at this juncture that I am advocating native app development exclusively, for all mobile application projects. I am not. I have described the native/web app distinction as a spectrum. This is truer today than ever before; the line between the two is becoming increasingly blurred. Facebook's new applications are not entirely native; the content is of a necessity delivered via the web, and some parts of the app are still built using HTML5, to enable regular updating. However, their apps are now considerably *more native* than they were before.

Some apps will always be more suited to web or hybrid development than others, and it is not always easy to know when this is the case. However, there are a few considerations that may help in the analysis; characteristics of apps, which may mean they may be suited to development as a web or hybrid app:

1. **Apps which have a lot of content that must be delivered via the web**. Some apps need regular or even constant content updates – Facebook and Twitter are good examples. In this case native development has fewer advantages, with respect to download speed and impact on a user's download quota. The app will need to retrieve data no matter how it is built. Furthermore, users will need to have an active connection to use the core functionality of the app. However, the lesson from the Facebook

story is that, even with an app that is primarily based on web content, it is risky to build the structures that display that content non-natively.

2. **Apps that need to be updated regularly and quickly**. Web and hybrid apps have much greater flexibility, and no one needs to monitor or approve content or structural changes, aside from the developers and content managers themselves. It should be noted, however, that the time taken for native app approval is not unduly long in most cases. At the time of writing, the average approval time for the Apple iOS App Store was estimated at 8.35 days.[3] Google's Play Store, for Android, takes a less-curative approach, and apps are often approved on the same day, sometimes within the hour.
3. **Apps with a customised user interface**. If you use a smartphone, consider the apps on it. In particular, consider the apps it came with pre-installed. You may not even think of these as "apps" - the phone dialer, the address book, the email client, the music player, and so on. These apps share common elements such as tab bars, buttons, content choosers and navigation structures. These apps, and the common structures, were built by the company which developed your device's operating system – Apple if it is an iPhone, Google if it is an Android phone, and so on. Now consider the other apps – the ones you chose to download. Some of those apps have similarities with the inbuilt ones – tools and structures where the developer has chosen to use the native APIs. Some will have less in common, and some will have nothing in common at all – they are said to have a completely custom user interface. Generally speaking, utility apps that have roughly similar kinds of features to the inbuilt apps tend to use somewhat standard interfaces (although many do not, and the number continues to grow). Games, on the other hand, tend to be entirely customised. As noted above, web or hybrid apps that try to replicate native user interfaces often do so poorly. Apps, like games, which have their own unique look and feel do not need to be concerned with this; they actually benefit from a consistent look from one device family to another, and a non-native approach to development may be entirely appropriate.
4. Apps which must be accessible to the widest possible number of users, and where this must be achieved within a limited budget. The various options for non-native development can make it feasible to develop for multiple platforms relatively cheaply, thus *theoretically* making it available to the largest number of potential users.

**Potential users are not users (yet)**

The italics in the last sentence are significant, and this serves as a segue into an important final point. It stands to reason that someone developing an application would typically want as many people as possible to use it. We place great significance on the size of the user base, taking it as one way of validating the project and justifying its expense. The app may have features that rely to some extent on the number of users, and of course if there is a commercial aspect, then the level of uptake will have a significant impact on the product's viability. In the health field, additional pressures may come to bear. The funders may require that the service be universally accessible to all potential users. While the goal of gaining as many users as possible is perfectly reasonable, here are two important things to consider.

1. **Smartphone users ≠ app users**. By "app users" here I mean people who are actively engaged with the third-party application ecosystem of their chosen device platform. People who are confident with the process of finding and installing apps on their device, who explore apps, act on recommendations to try this or that app, or generally have the inclination to wonder if an app exists for any particular need or problem they may have.

   Users of the different mobile platforms display very different patterns of use. A study in June 2012[4] compared the two dominant platforms, iOS (Apple) and Android (Google), and found iOS users to be 52% more likely to retain an app on their device than Android users. On average, 35% of iOS users launched an app more than ten times after downloading it, compared to 23% for Android users. Users of iOS also displayed a lower rate of one-time usage – instances where they installed an app, opened it once, and never used it again. Another telling statistic is the rate at which iOS users keep their devices up-to-date with the latest version of the operating system. Apple released iOS6, the latest upgrade to its operating system, in September 2012. In just the first week of availability, 100 million devices – around a quarter of those in use – had been upgraded. Fifteen per cent of devices were upgraded in the first 24 hours alone. The current install base is estimated at around 60%.[5] In contrast, at the time of writing Google estimates that just 1.8% of Android devices run the latest version, released in June 2012.[6] Running dated versions of the operating system limits the range of apps that can be installed. Prudent developers must ensure

their apps are backwards compatible, meaning that in many cases they must eschew the use of the latest and greatest features of the platform.

There are several reasons for this discrepancy. The iOS platform had a head start of several years, and became commercially lucrative for application developers in a relatively short space of time. This meant that a great many developers began, and continue, to develop for the platform, leading to a wide range of apps available for consumers. Apple's curative approach to application approval also meant that, at least initially, the quality of apps was relatively high – although the rapid growth of the platform, and the sheer number of new applications being submitted for distribution has led to something of a decline in the stringent enforcement of standards. This has led to a self-perpetuating cycle in the iOS app ecosystem. Users came to expect a wide selection of high-quality apps, therefore they became more likely to explore and use new apps, therefore the platform became more appealing for developers; rinse and repeat.

In contrast, the Android platform continues to grow in popularity, but has not yet achieved the kind of app ecosystem that exists for iOS. This may be a factor of the lower barrier to entry (leading to a comparatively higher proportion of low quality apps), the greater difficulty in monetising apps (it is much easier to download an Android app illegally, for free, than one for iOS) or the fact that Android developers have a much more difficult task optimising applications to the myriad devices they must support. Often the most realistic strategy is to take a "lowest common denominator" approach to ensure that apps work across the widest range of devices, running the widest range of operating system versions. This impacts on quality and perpetuates the problem of app engagement and commercial viability.

Finally, Android devices typically cost less and are more likely to be fully subsidised by the telephone carrier. As it becomes increasingly difficult to find a mobile phone that is not a smartphone, many people may find themselves owning an Android device almost by default. They simply wanted to purchase a new phone, have no intention to engage in app use or any other use aside from making calls and sending SMS messages, and were sold an Android device as the least expensive option (and often the one with the largest sales commission).

All of this adds up to the fact that, in estimating a potential user base, and deciding on platforms to support, it is important to not simply look at usage statistics in isolation. So, although Android recently reached parity with iOS in terms of Australian user base (38% of all mobile phones, compared with 37% for iOS),[7] it is not a given that half of a given app's user base will be on the Android platform. The picture is far more complex than this.

2. Everything is a trade-off. Even given the above, it may be tempting to think that there is nothing to lose in cross-platform development. Even if users on a given platform may be less likely to find and use the app, at least it is theoretically available to them. When you add in the fact that, in some cases, native development may be more expensive, it may almost be considered a "no-brainer".

   But once again, it is important to consider the implications. Developing an application capable of functioning on a wide range of devices requires compromises. Each of those devices has different specifications and capabilities, and non-native development will often involve the "lowest common denominator" approach described above, to deal with these differences. Application art – graphics, images, buttons and so on – may need to stretch to accommodate different screen sizes and resolutions, and this will typically lead to an inferior visual experience. Alternatively certain kinds of art may need to be avoided altogether. Developers will need to trade off performance and robustness – optimising the app for high performance may mean it is likely to be "buggy" on devices with lower capabilities, while opting for "safety-first" could mean the app has poor performance, even on higher-spec devices.

   These compromises may make an app less palatable for users, who have come to expect a high standard in mobile software. The danger is that, in trying to maximise your potential user base, you have effectively limited your reach by creating a compromised product. When Facebook delivered a substandard mobile app, hundreds of millions of people still used it because they wanted to access the service via mobile and, to them, subpar was better than nothing. It is fair to assume that health professionals looking to develop a mobile application do not begin with an existing user base of a billion people.

   Consider this analogy. You are developing a general-

purpose printed health resource. It is suggested that you use an extra-large type size, to make the material accessible for older people or people with limited sight. The argument is that doing this will maximise the number of people who may potentially access the information, and that "there's nothing to lose", because people with normal sight can still use it. However, doing this will involve compromises. If space is limited (say, in a brochure) images may have to be omitted, or reduced in size, to accommodate the text, and this may reduce the resource's impact. If it is not, the resource may become much larger, impacting on production cost and potential acceptability. If it is intended as a quick read for a GP's waiting room, and it appears to be very substantial, it may be less likely to be used. A decision in this case needs to be based on a clear understanding of the core target audience, the likelihood of various categories of people using the resource, and any other options for accessing the information.

My point here is that, although in general terms maximising accessibility is a worthwhile goal, it is not a simple calculation to make. If this is true for print resources, it is especially true for software, where so many more variables are at play.

**Conclusion (and, finally, the five questions)**

Mobile technology provides many opportunities for health professionals, service providers and health promoters. It is immediately accessible, is quickly becoming ubiquitous, and is increasingly turned to by Australians for all manner of information, including health information.

However, the process of contracting developers to build software can be fraught. Quotes can vary wildly – to the extent that it can be possible to wonder if different developers are proposing to build entirely different things. In fact, this is a pertinent question to ask, because this may very well be the case. Software development is not an "A+B=C" equation. When presented with a particular task (your application concept), a developer must choose from a multitude of different ways to accomplish that task, taking into consideration another multitude of variables. Not dissimilar to what a GP would have to do when asked the question, "Can you help me to be healthier?"

There are many questions that can assist in understanding and comparing proposals from different developers. Some are obvious, and can easily be understood by people with no experience in software development – what are the timeframes, how experienced are the developers, what hourly rate is being proposed, what guarantees are involved, who owns the final product, and so on. But in the case of mobile software, I have made the case that there are other important questions that relate to the core DNA of an application.

So, then, the questions.

1. Are you proposing to build this as a native or web application?
2. If it is to be a native application, is it genuinely native (i.e. written in a platform specific language) or cross-compiled (written using a third-party tool, then outputted for a range of devices)?
3. Which platforms will be supported, and why?
4. Why do you think this is the best approach?
5. What compromises will you need to make, in order to build using this approach?

Of course, answers to these questions will not provide a straightforward answer to the pivotal one – "which of these proposals should I choose?" But the responses should assist in weighing up proposals and making educated guesses at the potential quality, reach and appeal for a given application. There are no right answers, but some are better than others. Generally speaking, "that's the only way we know how", "that's the cheapest way", or "I don't really know" should be considered with caution.

If nothing else, having a conversation that goes deeper than the practicalities of the application at hand, and touches on the developer's underlying philosophy and approach to development, should provide insight into their knowledge and professionalism. Even if you do not understand everything they say, you will come away with a sense of the degree to which they "know what they're talking about". You may even gain some understanding of just what kind of shovel you're about to buy.

## References

1. Forbes. Facebook CEO Mark Zuckerberg: We Burned Two Years Betting On Mobile Web Vs. Apps. 2012 [cited 15 November 2012]. Available from: http://www.forbes.com/sites/roberthof/2012/09/11/mark-zuckerberg-we-burnt-two-years-betting-on-mobile-web-vs-apps/.

2. Hamburger E. Facebook for iOS goes native, waves goodbye to HTML 5. 2012 [cited 20 November 2012]. Available from:

http://www.theverge.com/2012/8/23/3262782/facebook-for-ios-native-app.
3. Shiny Development. Average App Store Review Times. 2012 [cited 22 November 2012]. Available from: http://reviewtimes.shinydevelopment.com.
4. Localytics. App Retention Increasing; iPhone Ahead of Android. 2012 [cited 15 November 2012]. Available from: http://www.localytics.com/blog/2012/app-user-loyalty-increasing-ios-beats-android/.
5. Chitika Insights. 60% of iPhones on iOS 6, iPad and iPod Touch Close Behind. 2012 [cited 15 November 2012]. Available from: http://www.zdnet.com/60-percent-of-iphones-now-running-ios-6-report-7000005169/.
6. Kingsley-Hughes A. Android 4.1 'Jelly Bean' reaches 1.8 percent market share. 2012 [cited 20 November 2012]. Available from: http://www.zdnet.com/android-4-1-jelly-bean-reaches-1-8-percent-market-share-7000005096/.
7.Marketing Magazine. Android dethrones iPhone as most owned smartphone platform in Australia. 2012 [cited 20 November 2012]. Available from: http://www.marketingmag.com.au/news/android-dethrones-iphone-as-most-owned-smartphone-platform-in-australia-17252/ - .ULMXd6XHYld.

## ACKNOWLEDGEMENTS

The author acknowledges the contribution of Mr. Adam Shaw, who provided technical advice on some aspects of this paper.

## PEER REVIEW

Not commissioned. Externally peer reviewed.

## CONFLICTS OF INTEREST

The author works as an application developer, and consultant, on a contractual basis.