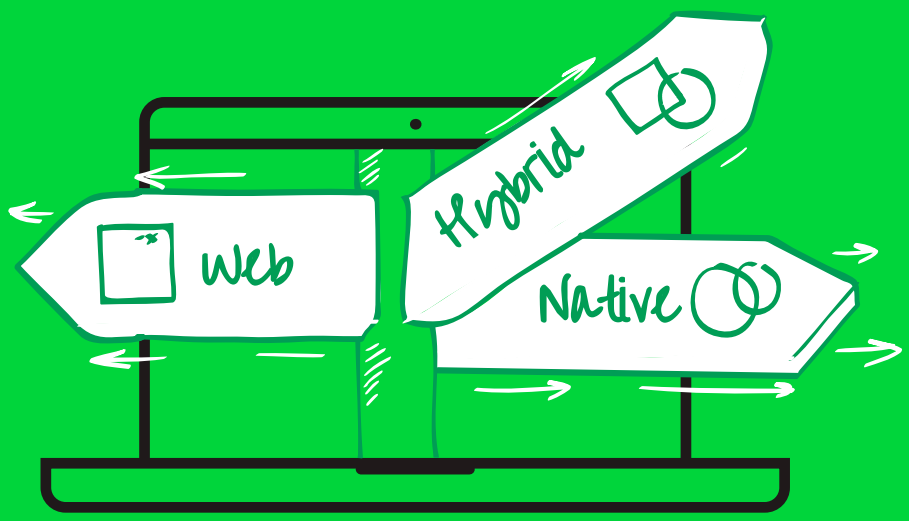


APPROACHING MOBILE

Understanding the Three Ways to Build Mobile Apps

Telerik

Mobile app development is complex.
To build apps that reach all users, developers must deal with many different operating systems, SDKs, development tools, screen sizes and form factors, as well as a technology landscape that is still in a constant state of flux. And if that were not enough, there are also several different ways to build mobile apps that development teams must sort through before beginning any new mobile effort.

Choosing how to build a mobile app, though, can have the most dramatic effect on the eventual cost, time, and success of a mobile project. This is second only to determining the scope and functionality of the app itself. Failure to match an application's requirements to the right mobile development approach all but guarantees wasted time and effort–often resulting in a less effective end result.

There are three primary approaches to building mobile apps today: Web, Hybrid and Native. This paper aims to explain the primary differences between these approaches, and provide a basic framework for choosing the “right” way to build modern mobile apps.

WHY ARE THERE DIFFERENT APPROACHES?

Mobile software development is still software development. If twenty years of “desktop” software development taught the industry anything, it is that every application is unique. Every application has requirements that drive the decisions about how it should be built. Some apps require maximum access to hardware for rich visual presentation. Some apps require maximum flexibility and the ability to quickly deploy changes in response to business needs.

A responsible software development strategy is built around a mixture of approaches that allow a business to cover all software requirements while optimizing the time and cost of delivering an app. Forrester Research shares the same belief in its study, ***Putting a Price to Your Mobile Strategy,*** suggesting developers not “get taken in by the allure of technology trends du jour.” The pattern for success is to realize each mobile app has a best-fit technology determined by mobile app objectives and available resources.

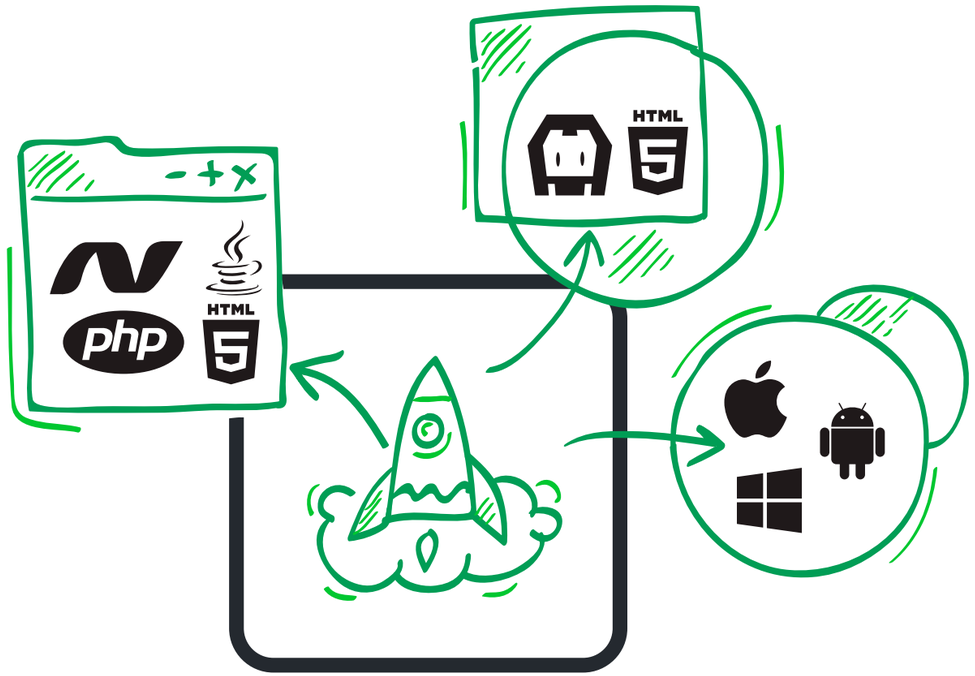
Before mobile development, it was well understood there are no silver bullets that solve all software development requirements.

A peek inside the software portfolio of any business today will reveal a mix of web-based applications, desktop-based applications, and perhaps software targeting cross-platform plugins, like Silverlight, Flash or Java.

> “A responsible software development strategy is built around a mixture of approaches that allow a business to cover all software requirements while optimizing the time and cost of delivering an app."

The same principles apply to a new era of mobile software development. A mature mobile organization with an optimized strategy will have a mix of mobile apps developed with Web, Hybrid, and Native approaches. The key is knowing how to choose the right approach for each application.

THE THREE PRIMARY APPROACHES



There are three broad approaches to developing mobile applications: Web, Hybrid, and Native. Each affords distinct advantages to a mobile development team, and none is a silver bullet that will meet the needs of all mobile applications.

As with all software development, there are tradeoffs that must be considered when choosing between these options. Understanding when and why to use each of these approaches is key to forming a consistent and optimized mobile strategy.

The following sections briefly describe each of the primary approaches to mobile development and highlight common pros and cons.

WEB

Mobile web development leverages the same skills and workflow traditionally associated with "desktop" web development. Developers build websites using HTML, JavaScript and CSS that are then accessed on mobile devices via mobile browsers. While some degree of local caching can be employed, most mobile web apps rely on a constant connection to the Internet and a web server to provide the views and content as a user navigates through the app.

There are two ways developers target apps for mobile devices with the web:

1. **Responsive Web Design:** With responsive web design (RWD), developers primarily focus on modifying the layout and display of existing desktop websites to adapt to the smaller screen size and touch-input of mobile devices. The advantage is a single web code base for both desktop and mobile users, but RWD is generally limited in its ability to create a "tailored" mobile experience that imitates the look-and-feel of native apps. It can also be challenging to use this technique when a desktop web experience contains complex widgets, such as data grids, that do not easily adapt to mobile screens with responsive CSS.

2. **Mobile Web App:** Alternatively, developers can build web-based experiences designed specifically for mobile users. In this scenario, mobile devices are usually detected and directed to a mobile optimized web app where developers can build tailored experiences that conform to mobile specific UI conventions. While much of a mobile web app's code base can continue to be shared with desktop web apps, this approach does require developers to build and maintain separate view (HTML/CSS) implementations for both mobile and desktop clients. Developers can choose to make mobile web apps look and feel exactly like installable mobile apps, or they can choose simpler presentations that feel more like traditional browser web apps (with no attempt to mimic native UI).

> Web is one of the most familiar and fastest ways to reach mobile users. "

Regardless of which approach to web a developer chooses, web is one of the most familiar and fastest ways to reach mobile users. No software installs (and subsequent updates) are required. Application access and information security can continue to happen in the data center on the server. And done properly, a mobile web app can reach all mobile devices with a browser, not just a limited subset of specifically targeted mobile platforms. Modern mobile browsers are even exposing an increasing number of device APIs, such as geolocation, via JavaScript, further enhancing the capabilities of mobile web-based apps.

That said, the trade-off for a mobile web app's familiar development platform and maximum reach is ultimately limited access to device capabilities relative to native and hybrid options. Unlike hybrid apps, which are able to expose essentially any device API via native plug-ins, web apps are limited to the features built-in to mobile browsers. Web apps also offer a very weak offline story in today's browsers, so any app that needs to work without an active Internet connection will be a challenge to implement on the web.

Even still, for apps that don't require offline support and don't exceed the capabilities of the web (no need to access device sensors/APIs, for example), the web remains a very compelling way to build and deliver mobile apps.

Essential skills: HTML, JavaScript, CSS

Essential tools: Anything capable of developing web apps

Platform reach: iOS, Android, Windows Phone or any HTML5 capable mobile browser

Sharable cross-platform codebase: 100% (UI + Logic)

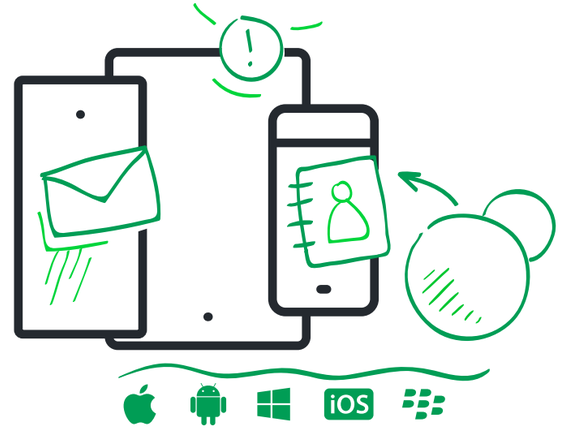
PROS

- Familiar, very low developer learning curve
- Easy to deploy, no software installs
- Easy to share code with desktop websites
- Maximum reach
- Reuse existing security and software management solutions
- Open standards-based platform (no vendor lock-in)

CONS

- Limited access to device hardware, APIs
- Poor offline support, requires "always on" Internet connection
- Unable to "install" on a device or publish via an app store
- Unable to match native performance for rich, animated interfaces

NATIVE



On the other end of the spectrum from mobile web app development is native app development. As the name implies, native apps are built using platform-specific SDKs and development tools provided by the platform vendors. For iOS, that means apps are built using Objective C in Apple's XCode. For Android, that means apps are built using Java and Google's Android SDKs. Windows Phone is .NET and Visual Studio, and so on. Every platform has its own SDKs, and often, its own programming language.

> If something can be done on a mobile device, then native apps will impose the fewest limits."

The advantage of native mobile apps, of course, is maximum access to the features and APIs available on each platform. If something can be done on a mobile device, then native apps will impose the fewest limits. As stated in an MGI Research study titled Buyer's Guide for Mobile Enterprise Applications Platforms (MEAP), "Native architecture tends to offer the richest, most graphically engaging user experience, high performance, and the ability to integrate with native device functions and back-end enterprise systems."

This power comes at the high cost of building apps that only reach one platform at a time, and with the requirement that development teams build and maintain multiple platform-specific code bases for the same app. Native app development is the most powerful, but the most expensive and slowest approach to reach all mobile users, especially if an app must support two or more platforms. Still, when maximum power is required, nothing beats native.

Essential skills*: Objective-C, Java, .NET, HTML/JavaScript

Essential tools*: XCode (for iOS), Eclipse (for Android), Visual Studio (for WinPhone)

Platform reach: Each app only reaches one platform

Sharable cross-platform codebase: 0% (No UI, No logic)

*Required skills and tools will vary depending on target platforms. To support the "top" mobile platforms today, a minimum of three programming languages and three IDEs is required.

PROS

- Complete access to device hardware, APIs
- Installable, can be app store deployed
- Maximum control over performance
- Powerful platform-specific development and debugging tools direct from platform vendors

CONS

- Multiple implementations required to reach multiple platforms
- Multiple skill sets and programming languages
- Requires installation (and device provisioning if private deployment desired)
- New tools needed to manage app security, enforce data security policies

Multi-Platform Native Development

Before discussing hybrid, it's worth mentioning a "middle-ground" option that has emerged for developing native applications across multiple platforms. With multi-platform native development, developers write an application in a single language (such as JavaScript or C#) targeting an abstraction layer to access native device APIs and SDKs. When the application is compiled, different app packages are produced that can run "natively" on different platforms. The result is a multi-platform "native" application written in non-native language, and a code base that can be largely shared between platforms.

The actual degree of "nativeness" and code reusability varies between multi-platform solutions, but in all cases the UI for the application is native. This is different from hybrid applications (discussed below), which rely on embedded web containers to present full-screen HTML-based UI. As a result, multi-platform native development is appealing when an app's UI complexity exceeds the limits of HTML, CSS and JavaScript, such as apps with lots of animation or movement.

The major drawback to multi-platform native solutions today is that the frameworks providing the abstraction are proprietary and reach a limited number of mobile platforms. Developers adopting one of these solutions will be "locked-in" to vendor specific abstractions (or bindings) and will be dependent on that vendor to continue to evolve the abstraction as underlying mobile operating systems change. In many ways, multi-platform native solutions face the same limits and risks as cross-platform plugins like Silverlight and Flash. If a company is willing to accept the risk of the underlying multi-platform technology, the solution can be a great path to simplifying normally expensive and time consuming native development for multiple platforms.

Essential skills: Depends on the multi-platform solution (examples: C#, JavaScript)

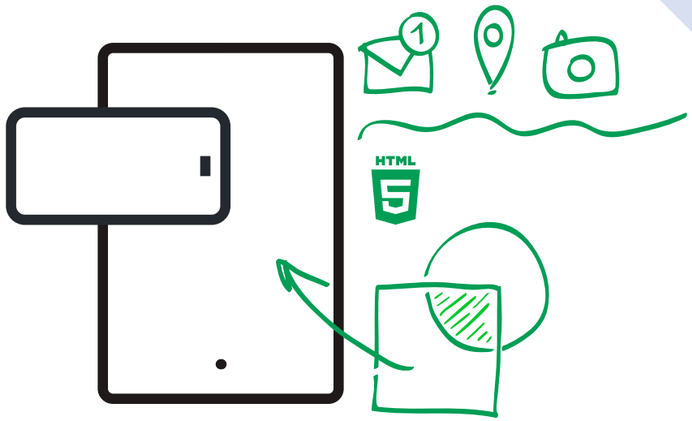
Essential tools: Usually a custom development environment

Platform reach: Limited to platforms supported by underlying compiler

Sharable cross-platform codebase*: Partial (UI is generally not sharable)

*Clearly, exact sharability depends heavily on the multi-platform native technology stack and app design. Some solutions provide better support for sharing UI and logic; some leverage platform-specific native UI, requiring custom UI implementations for each target platform.

HYBRID



Recognizing that most developers, given the choice, would prefer apps that have the “reach” of web and the “richness” of native, hybrid attempts to blend the benefits of web and native mobile app development. According to a recent Telerik Kendo UI Survey on HTML5, when asked what makes HTML5 development more appealing than other options for writing software, 62% said reach/cross-platform support as one of the biggest benefits. Hybrid apps are developed using standard web technologies, like HTML, JavaScript and CSS, but are able to overcome the limits of “pure” web apps by using platform-specific native “wrappers” to package and deploy the code. The native wrappers allow hybrid apps to be installed on devices, deploy via app stores and access native device APIs via JavaScript.

Since hybrid apps are built with web technologies, the learning curve is very low for web developers, and most existing JavaScript libraries can be leveraged from within a hybrid app. Seventy-two percent of developers surveyed by Telerik, noted the familiarity of languages as most appealing. Developers can further access any native API or device capability via plugins that expose additional native features to JavaScript code. Popular hybrid containers, like Apache Cordova, offer a rich ecosystem of available plugins, and developers with native programming skills can choose to create custom plug-ins tailored to specific app requirements.

Hybrid apps are completely self-contained. No server is required to launch or run a hybrid app, other than to supply or persist data within the app. In this way, hybrid apps are identical to native. When a hybrid app runs, the native application wrapper hosts a full-screen web container in which the HTML, JavaScript and CSS are loaded and run. To an end user, a well done hybrid app can be visually indistinguishable from a native app. In fact, prior to rebuilding their mobile apps natively, Facebook used hybrid technology to create some of the world’s most popular mobile apps, reaching millions of users. Most users, and even many developers, did not know Facebook’s apps were anything other than native. In fact, 52% of developers surveyed by Telerik in 2012 were unaware of this fact.

The primary limit of hybrid apps is the speed and performance of the web container on each target device. Older, slower devices require highly optimized code to achieve expected performance, while newer, faster devices are more capable of running more complex CSS and application JavaScript without making hybrid apps appear slow. For this reason, hybrid development is best used when the requirements of an app exceed the limits of web, but do not demand the full power of native. As an example, simple line of business apps that require offline support are great candidates for hybrid. Hybrid development for enterprise mobility is increasingly endorsed by leading industry analyst firms as well. In its 2013 release, Gartner recommends the hybrid approach, which “offers a balance between HTML5 and native” for Business-to-Employee mobile apps. Meanwhile, rich, interactive games or highly-animated interfaces are not good candidates for hybrid.

> “Hybrid development is best used when the requirements of an app exceed the limits of web, but do not demand the full power of native."

Essential skills: HTML, JavaScript, CSS, Hybrid container (such as Apache Cordova)

Essential tools: Anything used for web development* + hybrid SDKs

Platform reach: Limited to reach of hybrid container, but most reach all major platforms

Sharable cross-platform codebase: Almost 100% (Some platform specific UI may be desired)

*While web development tools can be used, tools designed and optimized for hybrid mobile development can improve productivity by helping with debugging, packaging and deployment to devices.

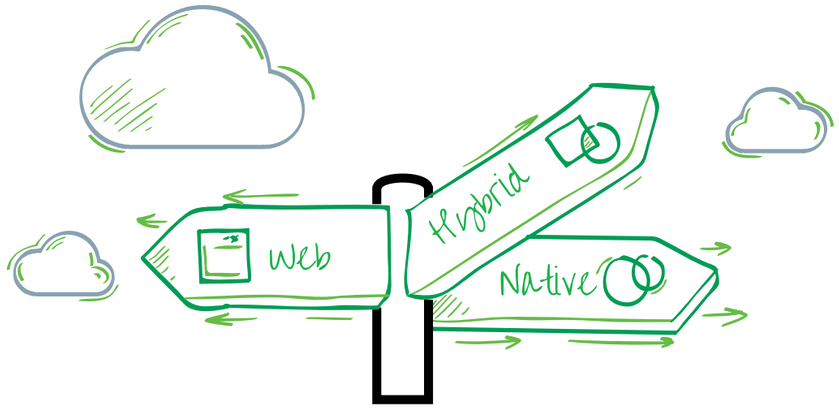
PROS

- Low learning curve for web developers
- Installed, can be app store deployed
- One code base for all platforms
- Easy to transition from web to hybrid development, reuse code
- Extensive access to device hardware, APIs

CONS

- Performance limited by web’s capabilities
- Requires installation (and device provisioning if private deployment desired)
- New tools need to manage app security, enforce data security policies

HOW DO YOU CHOOSE THE “RIGHT” APPROACH?



Choosing the “right” approach to develop a mobile app depends entirely on marrying the requirements and budget of an app to the capabilities and cost of a mobile development approach. As such, it is impossible to generically prescribe the “right way” to build every mobile app, but for businesses building many mobile apps, asking the following questions can help determine whether web, hybrid or native is right for any mobile project:

Who is the audience for the app?

There is a big difference between building internal Business-to-Business (B2B) or Business-to-Employee (B2E) apps and building public, consumer-facing, Business-to-Consumer (B2C) apps. The differences are similar to developing internal web apps or line of business apps and developing a company’s primary, consumer-facing .COM website or packaged, off-the-shelf software.

Internal apps often prioritize budget and flexibility over rich experience, making them ideal candidates for web and hybrid development.

How long do we have to develop the app—and for how many platforms?

Timelines have always been a critical factor in any software project. With mobile, if the goal is to reach multiple platforms, a tight timeline may require the use of web, hybrid or multi-platform native development. Developing the same app multiple times with native SDKs may simply exceed the time and budget of many projects.

If native is determined to be a requirement, plan on targeting one platform, such as iOS, first, and then rolling-out additional apps to additional platforms. In some cases, it even makes sense to build one native app for one platform, complemented by a web or hybrid app to address all other platforms.

What are the skills of our development team?

Teams with strong backgrounds in web development are going to be more productive faster with mobile web and hybrid app development. Strong skill reuse will minimize learning curves and often translate back into improved desktop web apps.

Meanwhile, if a team is not familiar with web development, or is not skilled in creating and debugging JavaScript, pursuing native or cross-platform native can avoid future frustration and wasted development time. The available native development tools can also help a team produce better results if they are not proficient debugging and optimizing JavaScript.

“Internal apps often prioritize budget and flexibility over rich experience, making them ideal candidates for web and hybrid development."

Does the app need to work offline?

If offline support is important, developing with the web will be much more difficult, if not impossible. Hybrid or native development will be needed to create the desired sometimes-connected capabilities.

Does the app need to access device APIs or hardware features?

While mobile web apps have access to basic device APIs and hardware sensors, apps that require more complete access should plan on hybrid or native implementations. If it is known up-front that an app will need push notifications, access to the device camera, contact list, file system or other native capabilities, you can quickly eliminate web as an option for developing the app.

What is most important for the app: Experience, Reach or Cost?

If maximum reach is desired, nothing beats the web. It spans all screens, even those not yet on the market today, while minimizing risk if popular platforms today fail in the future (such as BlackBerry). If maximum “experience” is desired, with rich, animated interfaces, native is the safest choice. Animation in particular is taxing on hybrid and web performance, and native will provide more polished results. Finally, if it’s a blend of reach and richness that’s required, hybrid is a solid choice that can help control costs and promote code reusability across platforms.

INEFFICIENCY OF "ONE SIZE FITS ALL" STRATEGY

It is tempting to look for the "one size fits all" solution to mobile development as a way of reducing complexity. Unfortunately, any strategy that attempts to develop all apps using one approach will ultimately force inefficient development decisions, waste development time and money, and limit the flexibility of an organization. Avoiding one or the other option completely is not wise either. As Forrester puts it, "Don't be a mobile technology lemming; simply because Facebook decided to move away from HTML5 doesn't mean that you should do the same."

For example, basic internal apps that need to quickly reach all employees and rapidly evolve with the business are likely best built as mobile web apps. No installs. Reusable code for desktop and mobile clients. The requirements do not exceed the limits of web, and the web offers a low cost, fast way to deliver the mobile solution.

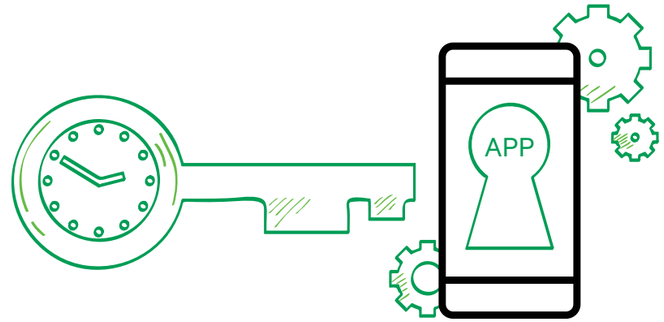
Meanwhile, polished apps that reach a business' customers via public app stores are better candidates for hybrid or native development. Consumers expect to discover apps in an app store. The app is less likely to change as rapidly. And overall, the goal is to optimize experience more so than cost.

> Any strategy that attempts to develop all apps using one approach will ultimately force inefficient development decisions, waste development time and money, and limit the flexibility of an organization."

If a "one size fits all strategy" is adopted, such as "all native" or "all hybrid" mobile development, inefficient decisions will be made in these scenarios. A company may overspend and reduce agility building many native internal apps, or it may fail to deliver the right experience for a polished consumer-facing app.

Beware the temptation to find a mobile development silver bullet, and instead adopt a smart, optimized strategy that draws on the benefits of web, hybrid and native app development.

MOBILE APPLICATION MANAGEMENT



One hidden implication of choosing between different mobile app development approaches is how that decision will impact the need for new application management policies and tools. Unlike desktop applications which tend to stay inside the firewall on computers that don't regularly leave the building, mobile apps are everywhere. They quite literally live with users, frequently traveling outside the boundaries of traditional corporate security.

This has significant implications on application distribution and security. When native or hybrid mobile apps are created, a company must deal with each of the following:

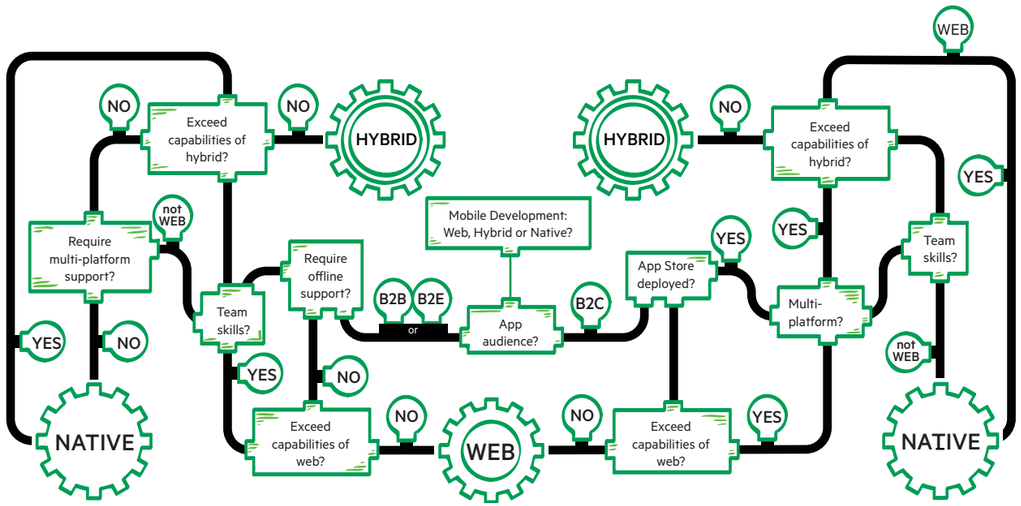
- How does the app get on "allowed" devices (assuming it's not a public app store app)?
- How is access to the app controlled (for scenarios where a device is lost or employee/partner access needs to be revoked)?
- How are updates delivered (and enforced)?
- How is cached data handled and secured on a device?
- How is app usage monitored? How are crash reports collected?

A company unprepared to deal with these challenges may have extra motivation to consider mobile web apps, which can typically reuse existing delivery and security policies set-up for traditional desktop web apps. With no installs and often no offline data storage, mobile web apps sidestep the need to introduce formal mobile application management solutions.

When choosing between web, hybrid and native, in addition to matching the strengths of each approach to a mobile app's requirements, a company should also consider the secondary impacts, like mobile application management, before making a final decision.

OPTIMIZED MOBILE STRATEGY DECISION GUIDE

The following diagram illustrates how to apply an optimized mobile development strategy to choose between web, hybrid and native for any project. Clearly, every organization will have different criteria to consider, but this example is a good starting point for any company building multiple mobile apps reaching different audiences, like employees (B2E), business partners (B2B) and consumers (B2C).



Conclusion

Mobile development may be complex and disruptive, but the core principles of software development that have been honed for more than twenty years still apply. Every app is unique, and that demands a strategy capable of intelligently delivering the right result. When a mobile strategy is built around the three approaches to mobile app development, every app can be delivered on-time and on-budget.

ABOUT THE AUTHOR

Todd Anglin
EVP of Cross-Platform Tools & Services

As the EVP of Cross-Platform Tools & Services at Telerik, Todd Anglin is responsible for the Telerik growing line of tools for web and mobile apps development, including Kendo UI and AppBuilder. He leads a global team of engineers, evangelists and business analysts and oversees the design, creation, sales and support of the Telerik industry leading HTML/JavaScript tools.

Todd joined Telerik in 2007 and prior to serving as EVP of Cross-Platform Tools & Services, he was Telerik Chief Evangelist, building and coordinating Telerik global evangelism efforts. Todd is a well-respected HTML5 industry leader and is an active member of the .NET and HTML5 developer communities. He is also a Microsoft MVP, founder and president of the North Houston .NET Users Group, and O'Reilly author.

Previously, Todd worked in Fortune 200 financial services enterprise IT, and has experience independently building and selling SaaS. He graduated with business honors from Mays Business School at Texas A&M University with a Bachelor's Degree in Business Administration. He is based in the Telerik office in Houston, Texas.

He is on Twitter at @toddanglin