Analysis of Native and Cross-Platform Methods for Mobile Application Development

Praveen Kumar S
Abstract
Brands today use smartphones and tablets to reach out to consumers. However, it is likely that mobile devices will evolve further with several new ones emerging in the future.

For mobile application (app) developers and related technology companies, the predominant question is – which device and related operating system should they target? While some developers evangelize native application development, others believe that cross-platform applications make more sense as they are device agnostic. This paper presents an analysis of both, native and cross-platform approaches, their features, advantages and disadvantages.

Introduction
In recent years, there has been a proliferation of mobile applications and the trend is likely to gather pace. That is a clear indication from Gartner’s new report, which states that by the year 2017, mobile apps will have been downloaded more than 268 billion times and generated a cumulative revenue of $77 billion. The surge in apps is also a result of numerous end-user devices entering the market. Apps should therefore be developed keeping in mind multiple devices and software distribution platforms, which cater to a range of screen sizes, hardware specifications, and configurations. Such complexity gives rise to a fundamental question for mobile application developers, “Which platform or technology makes the most sense and is suitable for the present and the future? Are native apps for a unique platform better, or is it preferable to go with platform-agnostic applications known as build-once-deploy-many (also known as cross-platform compilation)?”

This whitepaper looks at both, native and cross-platform tools to examine their utilities, features, advantages and disadvantages.

Native Application Development
‘Native’ is a term used for software development in which the developer uses the main language, tools, and a framework for the platform being targeted, while using an Integrated Development Environment (IDE). Native apps are typically built using development tools and languages (XCode and Objective-C for iOS apps, Eclipse, Android Studio; Java for Android, Visual Studio; and C# for Windows) that the respective platforms support, and they run only on those platforms. Since native apps are written for specific platforms, they can interact with and take advantage of operating system features and the other software programs installed on the platforms.
Features

- Multi-touch – double taps, pinch-spread, and other compound User Interface (UI) gestures
- Fast graphics API – extremely speedy graphics
- Fluid animation – crucial in gaming, highly interactive reporting, or intensely computational algorithms for transforming photos and sounds
- Built-in components – such as camera, address book, geolocation and other features native to the device. Another important built-in component is encrypted storage
- Interaction – interacts with other apps and provides for widgets on the homepage
- Flexibility – can respond to hard keys, i.e. the Android’s search button and volume control
- Documentation – there are nearly 3,000 books on iOS and Android development, along with several online articles, blog posts, and technical threads.

Advantages

Native apps are distributed through their platform’s native app store or marketplace. While there may not be a huge demand for business apps, app store distribution is essential for companies that require mass consumer distribution. It is important to note that hybrid apps also offer app store distribution.

Native apps offer the best graphics and animations, as they are built using the device’s native language and installed on the device itself. If a business needs a highly graphical application such as a game, native apps will be the best option.

Native apps have the benefit of familiarity. Most developers are already familiar with iOS and Android development kits and have a fair idea about how they really function. That simplifies the development process.

Native apps can remove authentication hurdles, and set up optimized and personalized experiences for end users.
While native applications offer benefits in graphics, app store distribution, and device integration, their lack of portability poses significant problems for businesses.

Besides facing the risks of an unstable mobile-platform landscape and limited app control, native applications require large investments in terms of time and money. While native app development costs vary according to complexity, it is definitely the most expensive and time-consuming approach. For example, Forrester Research estimates that most native apps require at least six months of full-time work, and cost between $20,000 and $150,000, depending on their complexity.

When placed in the app store, a native application is controlled by the app store’s owner (like Apple or Google). Thus, the app-store model places companies at the mercy of a third-party vendor. The challenge with this model is that the development cost and time to market are high, and organizations often have to compromise on the number of devices(s) they target.

There could be increased maintenance costs because native apps work in a silos-based model. Since each operating system is different, updates will need to be repeated for every application to ensure its compatibility with the device.

Lastly, since each platform has a specified process by which applications are approved, organizations will have to go through multiple processes to ensure successful deployment of their application on each device, making the process labor intensive.

### Limitations

- **Multiple OS Support**: No
- **User Interface Quality**: High
- **Performance**: High
- **Cost of Ownership**: High
- **Application Updates**: Native Market
- **Application Maintenance**: High
- **Development Languages**: Java, C, C++, Objective C, Objective C++

### Cross-Platform

- **Multiple OS Support**: Yes
- **User Interface Quality**: Medium to High
- **Performance**: Medium
- **Cost of Ownership**: Medium
- **Application Updates**: Native Market
- **Application Maintenance**: Medium
- **Development Languages**: Java, HTML, CSS, JavaScript

The total cost of ownership is higher for native apps.
Cross-platform app development empowers the developer to create an application using a single language or tool set, and instantly deploy it across a variety of platforms.

Cross-platform mobile development can involve either developing the original app on a native platform (which could be iOS, Android, Windows Mobile, BlackBerry/RIM, etc.), or developing the original app in a singular environment that will then allow the app to be sent to different native platform(s).

There are other methods by which cross-platform solutions are developed (for example: by compiling a native application, providing an interpreter framework, embedding web code (HTML/CSS) in a web browser-based app shell, or creating a simple mobile-enabled web app). Cross-platform evangelists firmly support their method, as it empowers applications to be modelled in an abstracted form and provides better user experience across multiple devices.

### Features

- **Familiar languages** – apps can be created with HTML, CSS, and JavaScript
- **Integration** – integrates with existing JSP and .NET infrastructure
- **Single code for building mobile apps for iPhone and Android platforms**
- **Advanced capabilities** – leverages features such as GPS, camera, etc.
- **Flexible** – applications adapt to different resolutions, screen sizes, aspect ratios, and orientations
- **Multiple devices** – can be used to build for desktop, tablet, and mobile web devices
- **Single-page architecture** – generates self-contained web apps that execute locally on the device.

### Advantages

The number of smartphone users is growing rapidly, with Apple’s iOS and Google’s Android occupying top positions. Developing an application that runs on both, iPhone and Android, is one advantage of tapping into the greater market potential.

Cross-platform simplifies marketing by enabling the use of multiple media with generalized messages targeting potential customers.
Maintaining and deploying changes when developing one single application running across all platforms is quicker, as updates can sync immediately. Further, it is much easier to manage a single team of developers working on a single multi-platform application than several teams working on different platforms.

The overall design and feel of the app can be maintained across various platforms if there is a single code running across the software.

When using the available tools, it is fairly easy to code in HTML5 and convert the same for different mobile platforms. This is a good method, as today, it is difficult to find mobile developers and relatively easier to find good HTML, CSS, and JavaScript coders. By using HTML5, hiring for development can be easier, especially for designing cross-platform apps.

Limitations

While cost saving can be one of the advantages of cross-platforms, the real meaning behind the term should be fully understood. A typical mobile application development process has cost overheads related to requirements gathering, analysis, and high-level design. It also has platform-specific design considerations, such as form factor, capabilities of operating systems, and hardware besides other similar concerns. In a cross-platform approach, the costs mentioned here remain largely constant, with some tendency to increase, as the savings are related only to a single code base.

Pertaining to single code base, if a particular issue is found and fixed, or a new capability/feature on one platform is added, the entire suite of target applications should be retested fully. This is a serious concern. The same is applicable even if the change is required for only one platform. The fact that the code is used for all platforms introduces a mandatory overhead – i.e. to test on each and every platform every time a change is ready to be submitted. Any change for a particular platform may have unforeseen effects on an unrelated platform.

Therefore, in essence, any real “cost saving” is debatable at best, and limited to simpler applications created on free cross-platform tools that are distributed to a smaller subset of platforms.

Application size also takes impact from the overhead of having to download the contents of the application (mainly consisting of the graphic and audio/visual components packaged in the app, as well as the code for the app). There is also the “runtime” component of the cross-platform solution, and the potential overhead introduced into the compiled code if the tool
As apps continue to play an important role in the business world, developers and organizations struggle to find the best development approach, but most of them realize that both approaches have their advantages and disadvantages. There is likely to be further fragmentation of mobile devices and technologies, all of which will play a huge role in escalating costs and time frames. At the same time, they will be adding to the complexity of the development process. There will also be more issues related to security, integration, and upgradation. There could also be new distribution channels, wherein developers can market their apps directly to consumers, instead of going through app stores. Social media and its power will continue to increase, and its effect will be palpable in the future mobile space.

Performance is a concern, and an experienced programmer will always be able to get better performance out of a specific platform when programming an app natively. Generic cross-compilers or run-time interpreters simply cannot make the same assumptions about what the app is trying to achieve.

Most cross-platforms provide for a capability mechanism. By using a native plug-in, it is possible to provide access to some functionality on a particular platform which cannot be encapsulated in the system. This highlights two concerns:

- The supplier understands that one cannot create a perfect system for all requirements, and therefore allows the supplier to 'plug-in' specific pieces of native code to resolve certain issues the supplier knows exists.
- Identifying people with the right skill set to create the plug-in code.

Many cross-platform solutions allow non-developers to use alternate skill sets (web designing, etc.) to generate an application. This is, in one way, similar to providing a set of tools that allow an accountant to repair a car, instead of hiring a car mechanic. While documentation may provide support and help to create simple solutions to common issues, it will not suffice if an issue crops up which requires specialized knowledge of the underlying platform or language.

Summary

As apps continue to play an important role in the business world, developers and organizations struggle to find the best development approach, but most of them realize that both approaches have their advantages and disadvantages.

There is likely to be further fragmentation of mobile devices and technologies, all of which will play a huge role in escalating costs and time frames. At the same time, they will be adding to the complexity of the development process. There will also be more issues related to security, integration, and upgradation. There could also be new distribution channels, wherein developers can market their apps directly to consumers, instead of going through app stores. Social media and its power will continue to increase, and its effect will be palpable in the future mobile space. Therefore, organizations should adopt a flexible approach, for which support, scalability, and integration become factors to consider. When that happens, decisions about the right platform or approach will fall into place.
While cost saving can be one of the advantages of cross-platforms, the real meaning behind the term should be fully understood. A typical mobile application development process has cost overheads related to requirements gathering, analysis, and high-level design. It also has platform-specific design considerations, such as form factor, capabilities of operating systems, and hardware besides other similar concerns. In a cross-platform approach, the costs mentioned here remain largely constant, with some tendency to increase, as the savings are related only to a single code base.

Pertaining to single code base, if a particular issue is found and fixed, or a new capability/feature on one platform is added, the entire suite of target applications should be retested fully. This is a serious concern. The same is applicable even if the change is required for only one platform. The fact that the code is used for all platforms introduces a mandatory overhead – i.e. to test on each and every platform every time a change is ready to be submitted. Any change for a particular platform may have unforeseen effects on an unrelated platform.

Therefore, in essence, any real “cost saving” is debatable at best, and limited to simpler applications created on free cross-platform tools that are distributed to a smaller subset of platforms.

Application size also takes impact from the overhead of having to download the contents of the application (mainly consisting of the graphic and audio/visual components packaged in the app, as well as the code for the app). There is also the “runtime” component of the cross-platform solution, and the potential overhead introduced into the compiled code if the tool does not adjust the generated code optimally. This may, in some cases, double the size of the app to impact down-time, potential data cost and finally, end-user experience.

Performance is a concern, and an experienced programmer will always be able to get better performance out of a specific platform when programming an app natively. Generic cross-compilers or run-time interpreters simply cannot make the same assumptions about what the app is trying to achieve.

Most cross-platforms provide for a capability mechanism. By using a native plug-in, it is possible to provide access to some functionality on a particular platform which cannot be encapsulated in the system. This highlights two concerns:

References:

- [https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options](https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options)
Headquartered in Santa Clara, California, Tavant Technologies is a specialized software solutions & services provider that leverages its expertise to provide impactful results to its customers across North America, Europe, and Asia-Pacific. Having leveraged its unrivalled capabilities and domain insights to create game changing results for leading businesses across chosen industry micro-verticals, Tavant Technologies is known for long-lasting customer relationships, engineering excellence and passionate employees. Founded in 2000, the Company employees over 1000 people and has been recognized as Top 25 Best Companies to Work For.

About The Author

Praveen Kumar S.
Technical Architect, Mobile Dev Project Team at Tavant

Praveen has a decade’s experience working on various domains including mortgage, gaming, and ed-tech. He has worked on several mobile application development projects for leading companies across the world. An experienced programmer, he has designed/architected reusable components in distributed environments.