WHITE PAPER

CHOOSING THE BEST MOBILE TECHNOLOGY

Overview

As business organizations continue to expand their mobile practices, finding a suitable mobile technology is vitally important. There are four common technologies to solve their business needs; vendor native tools such as Java and Objective-C/Swift, HTML5/JS with Apache Cordova, Classic Xamarin, and Xamarin.Forms. Each of these technologies is appropriate in certain situations, but not all. We took a look at which of these tools should be used and when. For the purposes of this evaluation, other tools such as Appcelerator and Telerik's Platform will not be discussed as they tend to be similar to some of the development platforms mentioned here.

Kevin E. Ford

Magenic® // *Software to move forward.™*

TABLE OF CONTENTS

Vendor Native Tools

Vendor native tools are provided by the mobile platform's creators and represent the vendor's vision on how mobile applications should be developed for the platform. In many cases these are referred to as native applications. Since they are made by the vendor they are tightly tied in the roadmap of the individual mobile OS and arguably create the most up-to-date and stable way of developing against it. Each vendor uses a different language. For iOS there is Objective-C and Swift, for Android there is Java, and for Window's Phone there is C#. When using the vendor native tools, there should be an expectation that there will be no front end code sharing across the different vendor's platforms.

iOS	Android	WinRt	Windows Phone
Objective-C/Swift/Xcode	Java/Android XML	• C# .NET/XAML • C++/XAML • HTML5/JS	• C# .NET/XAML • C++/XAML • HTML5/JS

Apache Cordova

Apache Cordova is an open source solution that comes in several flavors. Microsoft has Multi-Device Hybrid Apps and Adobe is a backer of PhoneGap; both using Cordova to power them. Cordova uses a native wrapper around custom built JavaScript code that is interpreted at runtime. This sort of application can access device specific APIs through custom built wrappers in the Cordova ecosystem. If access to an individual system's resources is not granted through the wrappers included with Cordova, custom wrappers can be created in the platform's native language.

The Cordova controls and code are a mixture of HTML and JavaScript. In this way it is very similar to a web application that is deployed and running locally. In general, if a UI is built once it will run and display the same way on all the different platforms. This can be very advantageous if the application needs to appear the same, regardless of where it is running. Since the code is JavaScript it is naturally non-typed and interpreted at runtime. A wide variety of third party JavaScript libraries are available to be used in Cordova applications.

iOS	Android	WinRt	Windows Phone
HTML5/JavaScript with a native wrapper			

Classic Xamarin

Xamarin apps are written in C# and compiled down to native binaries that run on the device. Generally, access is given to all system APIs through C# wrappers. If access to another component is needed, a new wrapper can be created in C#. Internally, Xamarin provides an implementation of the .NET framework that is used by developers.

Classic Xamarin uses native controls with a shared C# backend to provide platform specific applications. Normally, a UI would need to be built for each platform targeted. Xamarin uses native UI mechanisms for each platform (Storyboards in iOS and XML in Android). This can be very advantageous if the application needs to look like it is a native application on

"The technical problem that Xamarin is trying to solve is extremely complex; wrapping iOS, OSX, and Android platforms to create a unified way of accessing them through .NET languages."

the device, regardless of where it is running. The C# code is strongly typed and normally compiled before distribution. A wide variety of .NET libraries are available to be used in Xamarin Applications.

The technical problem that Xamarin is trying to solve is extremely complex; wrapping iOS, OSX, and Android platforms to create a unified way of accessing them through .NET languages. As the underlying native technologies change, Xamarin changes the wrapped APIs to compensate. While they have been good at getting day one support, this is not always so. For example, the iOS move to 64 bit was such a large change it took Xamarin a year to have a viable solution. With the underlying native technologies constantly changing for one platform or another, Xamarin must change its solution to react. Currently, Xamarin is a comparatively small company and the side effect of this constant churn has manifested itself in some bugs and resulting developer frustration.

iOS	Android	WinRt	Windows Phone
C# .NET/Storboard/XiB/ native controls in Visual Studio or Xamarin Studio	C# .NET/Android XML in Visual Studio or Xamarin Studio	C# .NET/XAML in visual studio	C# .NET/XAML in visual studio

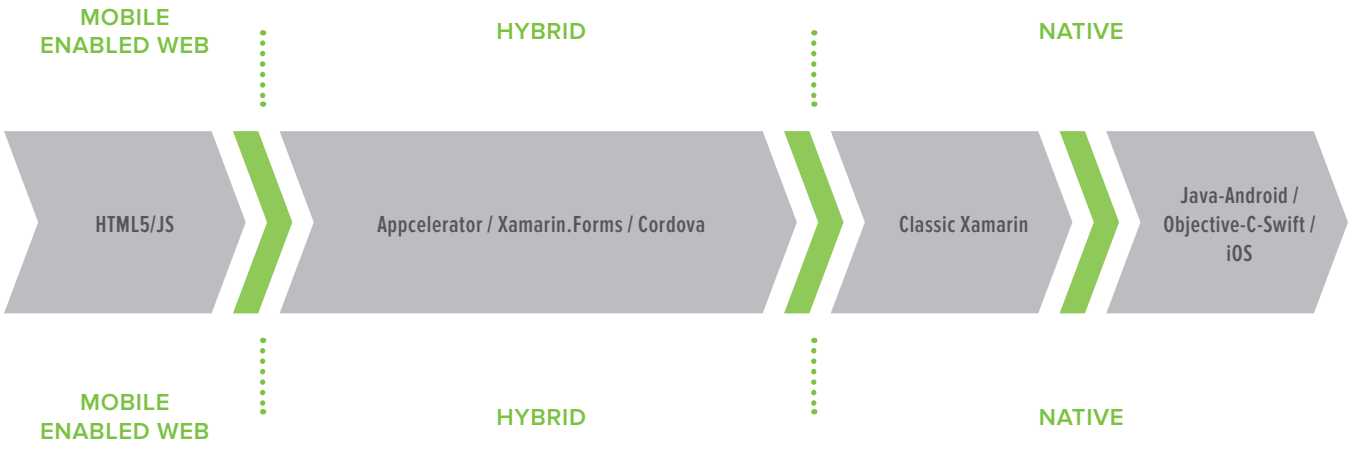
Xamarin.Forms

Xamarin has released a new technology called Xamarin.Forms. This allows an application to be written using a common UI technology, XAML, and work on the iOS, Android, and Windows Phone platforms while still using native controls and navigation schemes. Xamarin.Forms was written to compete directly with products such as Cordova when applications should appear mostly the same on all platforms while still retaining the speed, look, and feel of a native application. Xamarin.Forms sits on top of the Classic Xamarin technology stack and as such gains its benefits and drawbacks.

At this time, Xamarin.Forms UI technologies do not support Windows Store applications so a second UI would need to be built to target that platform. Xamarin.Forms can be used to create iOS phone and tablet apps, Android phone and tablet apps, and Windows Phone apps but not WinRT (Windows tablet apps). With Microsoft's push for unified Windows Phone/WinRT applications, this will likely change with Xamarin.Forms eventually being able to create tablet applications for the Windows platform as well. Currently, there is an open source project that partially extends Xamarin.Forms onto the WinRT platform.

iOS	Android	WinRt	Windows Phone
C# .NET/XAML in Visual Studio or Xamarin Studio with occasional platform renderers using native controls for non-out of the box functionality	C# .NET/XAML in Visual Studio or Xamarin Studio with occasional platform renderers using native controlsfor non-out of the box functionality	C# .NET/XAML in Visual Studio (XAML not the same, UI re-write)	C# .NET/XAML in Visual Studio or Xamarin Studio with occasional platform renderers using native controlsfor non-out of the box functionality

Cross Platform Solutions



Tool Strengths

Vendor Native Tool Strengths

- Because native apps run natively the apps run faster and smoother. Processing click and touch events are much faster as well, without noticeable delays.
- The UI is native and feels natural when using the app.
- There is a robust community of native application developers.
- Vendor Native Tools tend to be very mature and stable.
- Vendor Native Tools are tied into the OS's roadmap and tend to be up-to-date with all features of the OS.
- Objective-C/Swift/Java/C# are strongly-typed languages allowing bugs to be caught at compile time instead of run time, helping to minimize the number of defects introduced into the application.
- The tooling tends to be free with a large support community.

Apache Cordova Strengths

- Cordova development feels comfortable for front end web developers as it's simply HTML5 and JavaScript.
- Windows developers are able to use Visual Studio using the Multi-Device Hybrid Apps CTP. Emulator and builds for Mac are still possible using built in Visual Studio controls.
- Mac developers are able to use their standard tools without interference from Visual Studio files.
- Quick testing using Chrome and Ripple without having to build and deploy to emulators.
- Existing web components and code are reusable on Cordova. If the Cordova app is a replication of an existing website, then that website's code can likely be reused if well written.

"Don't just think about what you want your application to be like at launch, but where it will be throughout its expected lifetime."

- Cordova does not need multiple views built for different platforms. Once a Cordova application is built it can be deployed to iOS, Android, Windows Phone, or Blackberry with little or no changes assuming that the same UI is desired on app platforms.
- Cordova can share the same logic for all platforms.
- Cordova is a free and open source platform with a large community for support.

Classic Xamarin Strengths

- Classic Xamarin apps compile natively so the apps run like native apps. Processing click and touch events are fast as well, without noticeable delays.
- Classic Xamarin focuses on sharing backend logic, keeping the UI native, and feeling natural when using the app.
- Classic Xamarin apps can take advantage of many of the third party libraries written for the .NET stack including retrieving components through Nuget and the Xamarin Component Store.
- The Xamarin development style feels natural to .NET developers.
- Xamarin can potentially reuse much of the server side business logic if needed for offline capabilities.
- C# is a strongly typed language allowing bugs to be caught at compile time instead of run time, helping to minimize the number of defects introduced into the application.
- Xamarin has strong support for advanced development tooling and process control such as unit testing, mocking, dependency injection, and inversion of control.
- Backend code can be written in C# against the .NET framework allowing code to be shared across platforms and from server to client even in connected scenarios.

Xamarin.Forms Strengths

- Xamarin.Forms apps compile natively so the apps run like native apps. Processing click and touch events are fast as well, without noticeable delays.
- Xamarin.Forms focuses on sharing backend logic and UI logic. The XAML is converted into native controls so the UI is native. This can feel natural when using the app.
- Xamarin.Forms apps can take advantage of many of the third party libraries written for the .NET stack including retrieving components through Nuget and the Xamarin Component Store.
- The Xamarin development style feels natural to .NET developers, particularly if they are used to the MVVM pattern.
- Xamarin can potentially reuse much of the server side business logic if needed for offline capabilities.
- C# is a strongly typed language allowing bugs to be caught at compile time instead of run time, helping to minimize the number of defects introduced into the application.
- Xamarin.Forms has strong support for advanced development tooling and process control such as unit testing, mocking, dependency injection and inversion of control.
- Backend code can be written in C# against the .NET framework allowing code to be shared across platforms and from server to client.
- When required, developers can use native UI controls and renderers if the functionality for a given platform does not feel right given the out of the box functionality.

Tool Weaknesses

Vendor Native Tool Weaknesses

- When writing for two or more platforms, such as iOS and Android, all client side code will need to be replicated as different languages are at play.
- It is very likely that the client side technology will be different than the server side, resulting in business logic that exists on both the client and server to need to be rewritten for each platform.
- There is a high need for platform-specific knowledge on the part of the developers.
- Some of the platforms, particularly iOS, recommend a very rigid way to think about application development which on occasion leads to very in the box thinking about how the application can be constructed.
- Since the industry is at an inflection point, it is unclear what the mobile world will be in several years. Vendor-specific tooling may lead to investment in a knowledge that may become less relevant in the future. This is particularly problematic of languages that exist for no other purpose than to support a given mobile platform.

Apache Cordova Weaknesses

- Cordova has a weakness in terms of execution speed on the device. Testing we have conducted found that in many cases once the application was up and running it was not significantly slower than with native compilations such as Xamarin, however the initial application start time was about twice as long on both iOS and Android.
- Cordova is powered by JavaScript so it is not a complied language and therefore does not gain the benefit of build time type errors. This can be partially mitigated by using tools such as TypeScript.
- JavaScript, and by extension Apache Cordova, do not have rich unit testing, mocking and inversion of control tools. Tools such as Jasmine exist but are comparatively immature. This can add to overall total cost of ownership of the application.
- By default, HTML touch handling adds a .3 second delay. While this can be turned off in some cases, it is not always possible with all libraries. We found in certain cases we could not turn off the delay without causing problems and it caused a noticeable user experience degradation.
- Cordova applications that need significant offline or business logic contain heavy use of JavaScript that is more common among Single Page Application (SPA) frameworks like Angular. While there is nothing wrong with this, there can be a learning curve for developers who are more used to server side web technologies such as ASP.NET or ASP MVC.
- When the UI is expected to be different across platforms, Cordova loses some of its advantages for code sharing. There are tools to make it look correct on the different platforms but these are not without their own development and maintenance overhead. These tools can also lag behind the look and feel of the platforms as they are upgraded.
- Most platform-specific features are allowed through the use of wrapping libraries. However, when trying to access a feature not yet implemented the developers may have to write wrappers in the Vendor Native Tools. Additionally, some of the existing wrappers

are immature or incomplete. For example, we found in testing that the wrappers for accessibility were problematic and didn't always work correctly.
- There is always the possibility that the device vendors will disallow applications written in this technology, either through technical means, store control or licensing.

Classic Xamarin Weaknesses

- Developers need much more platform-specific knowledge than when using a tool like Cordova.
- Developers normally need to be connected to Macs.
- Some features need to be done on a Mac (Xib) in Xamarin Studio instead of Visual Studio in Windows.
- Apple restrictions limit the parts of the .NET framework that can be implemented (JiT in particular).
- Classic Xamarin is adept at creating applications that are native to the platform. There is extra overhead in creating applications that look the same on all platforms.
- Native Android UI development requires a lot of per platform tweaking. While there are less form factors for iOS, it exists for iOS as well.
- While Classic Xamarin does compile to a native application, tools that expect the binary to look a certain way may have issues. We have found some MDM applications that wrap and modify the binary have issues with the structure of the compiled binaries, even if the actual devices do not.
- Classic Xamarin does not have the same level of community participation, knowledge and support as either the Vendor tooling or HTML5/JavaScript (Cordova) development.
- There is always the possibility that the device vendors will disallow applications written in this technology, either through technical means, store control, or licensing.

Xamarin.Forms Weaknesses

- Xamarin.Forms is a relatively new technology that is still in a high state of flux. This means that development outside of a relatively narrow look and feel will lead to platform-specific code. Additionally, there may be significant time spent dealing with updates and bugs.
- At the time of this writing there is currently no UI designer for the XAML used by Xamarin.Forms. Also in Visual Studio there is no XAML intelligence (it exists in Xamarin Studio). This can lead to extra development cycles trying to get the UIs to display correctly on all platforms.
- There is currently no support for Windows Store Applications (WinRT) and the XAML is different than the XAML used by this style of application. Any application that will span Xamarin.Forms mobile and Windows Store applications will require the UI to be written at least twice. There is an open source project that provides partial support and we expect Xamarin.Forms will be extended to the WinRT platform eventually.
- Many of the current APIs are protected as Xamarin has not released them for public consumption. This may lead to problems when trying to override the default behaviour of Xamarin.Forms. This is particularly problematic as the default behaviour can still be quite limiting and there are bugs that need to be worked around.
- Developers normally need to be connected to Macs.

- Apple restrictions limit the parts of the .NET framework that can be implemented (JiT in particular).
- While Xamarin.Forms does compile to a native application, tools that expect the binary to look a certain way may have issues. We have found some MDM applications that wrap and modify the binary have issues with the structure of the compiled binaries, even if the actual devices do not.
- Xamarin.Forms does not have the same level of community participation, knowledge, and support as either the Vendor tooling or HTML5/JavaScript (Cordova) development. Currently it has the lowest level of community involvement of the four options evaluated just due to the relative newness of the technology.
- There is always the possibility that the device vendors will disallow applications written in this technology, either through technical means, store control, or licensing.

Factors to Consider

UI Design

The design of the user interface itself can be one of the largest determining factors in what technology to use. If it is desired that the UI look the same on all platforms then Cordova has a significant advantage over all the other technologies mentioned here. In such a case it truly can be near to write once and run everywhere.

In general this type of UI for mobile devices is not recommended by Magenic. The normal usage pattern for users is that they run multiple applications on the same device and expect all the applications to work the same on that device. Applications that work and look the same regardless of platform are more appropriate for situations where users switch from device to device and need the application to look the same on all of them. If it is the case where individual users generally use the same devices all the time then a design that matches the platform may be more appropriate and yield higher productivity. For this a native UI (vendor native tools, Classic Xamarin or Xamarin.Forms) may be a better choice. However, if it is indeed the case that application users move from device to device, platform to platform frequently, then using the same non-mobile design for all platforms would be more appropriate and thus Cordova.

Generally the controls used and capabilities of a particular UI create a framework that should be considered in the design of the application. If the design follows the intentions and capabilities of a web page, then HTML will likely be a much more efficient tool in creating that UI than native Android xml or iOS controls. For example, a multi-level showing and collapsing of sections of a screen is a common web design pattern that is well suited to the capabilities of div tags. Trying to make Android xml or native iOS controls work this way is fighting how these technologies are designed to work. Conversely, both Android xml and native iOS controls have native ways to manage long lists and navigate them. A design that assumed this functionality would create extra work with straight HTML.

See the following for a good take on UI consistency and where it is appropriate:

http://connorsears.com/2012/08/12/consistency

This factor will be more important for instances where the UI is the main part of the application with little or no business logic.

Note: *an argument can be made that by making the UI the same on all devices, it will be easier to support. While this is true, it is overshadowed by the ratio of users to support personnel. That is to say, making the applications more productive for the users of the application will likely yield increased organizational productivity over making the UI of the application easier for support personnel.*

Industry Trends

The computing industry has been in the process of undergoing an inflection point ever since the introduction of the iPhone. This has created disruption in what devices we use to do our work, how we interact with them and how we develop applications to take advantage of that. The future of the platforms and tools is unclear and worse than what we experienced in the early 1990s.

The inflection point means that it is impossible to accurately predict the future of the industry. Will one platform gain supremacy and if so which one will it be? Will no platforms gain supremacy and thus will we remain in a multiple device world? This uncertainty brings risk to any technology decision that impacts this space. If indeed one platform emerges dominant within the lifetime of the applications then many of the advantages of using cross-platform tools like Cordova and Xamarin evaporate.

There is even risk in developing for today's form factors. Those are still in flux as well with the emergence of wearables and multi-factor devices that may further change how we interact with our applications. Having said that, while organizations need to keep their vision looking toward the future, they must also do business today. That means creating a strategy to deal with today's environment of mobile devices.

Corporate Support

Vendor Native Tools

Apple, Google, and Microsoft are committed to their native development stacks and have the cash reserves to weather most short and medium term industry trends. However, given any protracted shifts in market direction, even large companies will change their strategies and focus.

Apache Cordova

Apache Cordova has large backers in Microsoft for Multi-Device Hybrid Apps and Adobe for PhoneGap but the underlying core is open source. This always carries with it the risks associated with any such product. While the source code is freely available to maintain, it is a task that most company's internal IT departments likely have neither the skills nor bandwidth to do on their own. That means that the viability for enterprise apps for Cordova in the future partially rests on the support of companies like Microsoft and Adobe and also the continued interest by the community that maintains and enhances it.

Xamarin

Compared to companies such as Google, Apple, and Microsoft, Xamarin is a very small player. This always brings into question if they will be able to survive in the long term. This will continue to be a risk but they do have support from companies like Microsoft who mentioned them extensively at the recent 2014 Connect event. While Microsoft didn't purchase Xamarin, they have partnered with them on several initiatives like the .NET Foundation. Additionally, their ability to deliver day one support for different iOS platforms indicates that they have support from other vendors as well.

Microsoft has moved parts of the .NET framework to open source and has announced that it is working on a cross-platform version which will encompass iOS and Android, partnering with Xamarin to make it happen. Microsoft is also increasing their DevOps capabilities by creating a cross-platform build agent. With increased Microsoft support, particularly with the planed cross-platform version of the .NET Framework, it is likely that some of the stability problems will be mitigated in the future.

Xamarin recently received $54M in round C VC founding that should provide cash flow for their organization for the immediate future. There is always the possibility in the future they will develop cash flow issues or key partners like Microsoft will turn away from them or duplicate their efforts reducing the viability of their product. However, it is obvious that Microsoft is currently committed to the Xamarin technologies and as such we believe they are a safe bet to create enterprise applications from an ongoing support perspective.

IT Context

TCO

Cordova has some advantages in initial development time when used in a situation where the UI is the same across devices and not meant to look native on the platform. The cost of the licenses is also free. From these perspectives it can be an attractive choice when targeting multiple platforms. These factors can be offset by the immaturity of development tooling such as unit testing, inversion of control offerings, and the high level of flux that is currently being experienced by JavaScript libraries in general. Initial cost is an important factor, but it should be weighed against the cost of the solution for its lifetime.

Xamarin does come with an associated tooling licensing cost that may not be borne by the other options. However, when looking at the entire cost of initial development and lifetime maintenance on a larger mobile application, this is likely not a significant amount. Magenic believes the cost of licensing should certainly be considered but not to the exclusion of other factors. Factors such as application user productivity and development time usually far eclipse the licensing cost. This is definitely a case where a more in-depth analysis should be employed to avoid a "penny wise, pound foolish" outcome.

Developer Skillset

An important consideration when looking at this from an IT context is who is going to create and maintain the application over time? Do they have the internal skills and knowledge to write and maintain applications primarily written in C# or do they have the skills and knowledge to write extremely JavaScript intensive applications? If this application will be maintained by your company, a good indicator would be to look at the types of applications you are building and maintaining today. If you are primarily writing C# client side applications or server side heavy ASP.NET or ASP MVC applications, it is likely that you will be able to move into a Xamarin environment much more easily. However, if you are primarily working in JavaScript frameworks such as Angular or writing other client side heavy web applications, you may be able to move into Cordova in a more cost effective manner.

Client to Server Code Sharing

Additionally, since these applications will need to work offline, it is likely that there may be opportunities for logic to be shared and maintained on the client as well as the server. In order to best capitalize on code sharing opportunities between the client and the server the same language would need to be used in both locations. That means using Xamarin if the server side technologies are writing in a C# technology or using Cordova if the server side code is in JavaScript using Node.js or similar.

Number of supported devices

The Android ecosystem has a huge number of potential devices, capabilities, and screen resolutions. The impact of this fragmentation was experienced even when trying small applications on different Android devices. This is also increasingly becoming a factor in the iOS ecosystem. For a good view of the potential for fragmentation in the Android and iOS ecosystems see the following site:

http://opensignal.com/reports/2014/android-fragmentation

Dealing with different resolutions and screen sizes has been something that web applications have experienced for a long time. Responsive web design tools have been built specifically to address this problem. We have found that Cordova's use of responsive web technologies made it much more tolerant of different device types than a Classic Xamarin or vendor native tool implementations.

Xamarin.Forms using XAML has been designed with responsive technologies in mind. However, Xamarin.Forms does not have the years of stabilization and work that exist in the HTML5/JavaScript/CSS technology stack dealing with this problem. Generally, the more different types of devices that will be supported, the more attractive the responsive UI technologies employed by Cordova, and to a lesser extent Xamarin.Forms, become.

Normally when more than two platforms are targeted, the attractiveness of maintaining more than one native UI quickly diminishes unless there are other compelling reasons to do so. There is, at minimum, an incremental increase in maintaining each UI. If there is a desire to support Android, iOS, Windows Phone, and Windows Store then Cordova and Xamarin. Forms become more attractive solutions than Classic Xamarin or vendor specific tooling.

Offline Use

In some cases mobile applications need to work when disconnected from the server. This means that data will need to be validated and stored on the local device when offline. It is likely that much of the validation and authorization logic will be the same for the application if it is online or offline.

Depending on how the server side code was written (i.e. written in a modular way, inversion of control, etc.) some of this logic may be reusable on the client as well as previously mentioned. This would only be feasible if the client and the server share the same language. In terms of Cordova this would mean that the server side services are written in JavaScript using a technology like Node.js. For Classic Xamarin and Xamarin.Forms this would be possible if the server side code were written in C# using something like Web API. When targeting multiple platforms using vendor native tooling, this type of logic sharing is not possible across platforms.

The importance of this factor depends on if the server side code is written in JavaScript, C#, or Java; how much business logic needs to be the same on the client; and if server side code that is already written was written in such a way that it can be shared.

If the business logic code were written with separation of concerns principles being followed then there is a greater chance of reuse. If separation of concerns was not followed, then even if the business rules are already in the same language, they cannot easily be reused. Business rules that are deeply embedded in other code (ViewModels and Controllers for example) cannot be easily extracted for use within either framework. In this case, it may be easier to rewrite the business rules which would negate some of the benefit of having the business rules in the same language.

Performance

Performance timings we have done show that there are a few key areas where there is a noticeable difference between native compilations like Classic Xamarin/vendor native tools and Cordova. Differences are particularly noticeable in load times. Cordova packages are also larger which would lead to somewhat longer application download and update times. Xamarin.Forms is an additional abstraction layer on top of Classic Xamarin and we have noticed degradations in performance in certain circumstances.

How much this factor is a determinant depends on how sensitive the user base is to these issues.

Decision Matrix

Note: these individual rankings can and should be influenced by the interactions of all the factors

FACTOR	VENDOR NATIVE TOOLING	APACHE CORDOVA	CLASSIC XAMARIN	XAMARIN FORMS
NUMBER OF PLATFORMS				
Single Platform	+ +	–	+	–
Two platforms	o	+	+	+
Three or more platforms	–	+ +	o	+ +
UI DESIGN				
This factor is more important when there is very little business logic, a complex UI and/or three or more different platforms targeted.				
The UI is designed as a web application, assuming HTML capabilities and not particularly following mobile platform standards and looks exactly the same on all platforms.	– –	+ +	– –	–
The UI is designed to follow platform standards / look / feel and generally follows platform specific navigation with moderate use of non-standard controls and patterns.	o	o	o	+
The UI is written following platform standards / look/ feel and generally follows platform specific navigation with light or no use of non-standard controls and patterns	+ +	–	+ +	+
TECHNICAL DIRECTION				
The importance of the skillset of the internal groups that will be creating and maintaining the application.				
Applications will be created and maintained by a third party	o	o	o	o
Applications will be maintained internally by a group that currently has primarily HTML5 / JavaScript skills	– –	+ +	– –	–
Applications will be maintained internally by a group that currently has primarily XAML / C# skills	– –	–	+	+ +
Applications will be maintained by groups that currently have native Android / iOS skills	+ +	–	+	o
Applications will be maintained by groups with mixed HTML5 / JavaScript and XAML / C# skills	– –	+	+	+

Highly Advantageous: + + **Advantageous:** + **Neutral:** o **Disadvantageous:** – **Highly Disadvantageous:** – –

Decision Matrix (continued)

Note: these individual rankings can and should be influenced by the interactions of all the factors

FACTOR	VENDOR NATIVE TOOLING	APACHE CORDOVA	CLASSIC XAMARIN	XAMARIN FORMS
NUMBER OF SUPPORTED DEVICES				
How many different device types are planned to be supported by the organization. This factor will be impacted by the type of UI selected.				
The organization plans at some time to allow any device and platform through BYOD	–	+ +	–	o
A finite number of devices will be allowed on several platforms	o	+	o	+
A small number of devices on a single platform	+ +	o	+	o
SERVER SIDE CODE SHARING				
The impact of the amount of business logic that can be shared from client side to server side. How the server side services were written will dictate if this type of code sharing is possible.				
Due to different languages, requirements or design no code can be shared from server to client applications or no desire to have complex logic on the client	o	o	o	o
Business logic can be shared and server side services are written in JavaScript (Node.js) and a fair amount of logic needs to be shared	– –	+	– –	– –
Business logic can be shared and server side services are written in C# (WebAPI or similar) and a fair amount of logic needs to be shared	– –	– –	+ +	+ +
PERFORMANCE				
The importance for the application to perform well in all circumstances.				
Performance has little or no impact on users as long as it performs "good enough"	o	o	o	o
Users are highly sensitive to the application performance	+ +	–	+ +	o
STABILITY				
The stability of the toolsets.				
The stability of the toolset is extremely important	+ +	+	–	–
DAY ONE API SUPPORT				
The importance of access to the latest platform capabilities				
Access to the latest platform specific capabilities is extremely important	+ +	–	o	o

Highly Advantageous: + + **Advantageous:** + **Neutral:** o **Disadvantageous:** – **Highly Disadvantageous:** – –

Final recommendations:

There is no single silver bullet that can decide what technology is best for you. In fact, different technologies are appropriate in different situations. We have found that the design of the UI itself can be one of the most significant factors in terms of development time. If the UI is designed in such a way that it should look the same on all platforms and basically follow HTML/css paradigms then the advantages experienced by Cordova will be nearly insurmountable to overcome. However, as soon as that equation starts changing where the UI will match the platform but have the same flow or a different flow, then some of the other options start to become much more attractive.

Vendor native tooling is, in many respects, the least risky. The Xamarin and Cordova solutions are in one way or another built on top of the vendor native tooling and thus add extra complexity to the solution. The vendor native tooling is more focused on their individual platforms and will be the first implementer of new platform capabilities. As long as a given platform survives it is likely that the vendor native tooling will as well.

Having said that, we believe that an all-inclusive view is best in determining a solution. What are the capabilities and technical direction of the groups that will create and maintain the applications, what platforms are you targeting now and in the future, who is using the app and how, what are their expectations for performance and experience, what do you expect the total cost of ownership to be (don't just focus on licensing costs but also how UX and user productivity can impact the organization's TCO), and what do you want the application to look like? Don't just think about what you want your application to be like at launch, but where it will be throughout its expected lifetime. By weighing all of these factors a reasonable conclusion that can be reached to provide the best solution.

Why is this so hard? Because we are in an inflection point, the future is unknown, and multiple solutions are popping up to handle the different potential outcomes. If you had a crystal ball that can see the future of mobile then some of these possible solutions would become much more attractive. In fact, some people claim to know the direction of the industry and therefore which tool you should choose but we are cautious of such claims. At this time Magenic believes that the inflection point is still happening and thus the future of the mobile industry is unknown. As such, we suggest looking at factors you can know or at least predict with a reasonable level of confidence and make your choice based on those.

About Magenic

Founded in 1995 by the same technical minds that still run the company, Magenic delivers software results on the most complex modern application projects.

Visit us at **magenic.com** or call us at **877.277.1044** to learn more or to engage Magenic today

Magenic® // *Software to move forward.™*