Xamrin – Future of Mobile development

Xamarin Forms is an abstraction framework from Microsoft which allows us to reduce the amount of platform specific UI code required when creating cross platform apps. Forms builds upon Xamarin's existing technologies: Xamarin.iOS and Xamarin.Android, and also allows for compilation to windows platforms. UI's can be created in C# or using Xamarin's own dialect of the XAML markup language. The idea behind Forms is to allow developers, especially those in the enterprise, to further increase the amount of code-reuse between platforms. Already, with Xamarin.iOS and Xamarin.Android, one can see large amounts of code-sharing between platforms, especially when engineering apps appropriately (using MVVM pattern, or similar abstractions, correctly separating concerns, data and service layers).

How does Xamrin work?

Xamrin apps follow a similar pattern as that taken by MvvmCross (which is a project forked from MonoCross, preferring MVVM to MVC). This is to say that the platform provides view elements (containers, controls, views), which are used in shared projects (typically PCL libraries). The actual rendering of these views is handled by a "renderer" class, which acts as a "presenter" for the core (shared) view classes. Renderers are written per-platform. In this way, Forms can allow a developer to use general abstracted views (such as Entry, or Button), which are then rendered and behave natively on each platform.

What are the benefits to this approach?

For projects which require multi-platform presence there are many benefits:

Code sharing between platforms is further increased, as the view layer code is now also shared. The "look" and (more importantly) feel of the application fits the target platform, due to using native views and controls – the performance is therefore also native. The platform is extensible and

configurable: developers can extend existing controls to provide native features that Xamarin didn't include out of the box, add new features, or workaround bugs. Performance issues are mitigated, as the developer has more options for resolving performance issues. Access is available to native features of the device. Ability to integrate with Xamarin.Android and Xamarin.iOS controls and classes. All the other benefits of Xamarin platform (apps are written in C#6 across all platforms, Nuget packages, excellent debugging, Xamarin's additional features such as TestCloud, great documentation, support, community, etc.) Many containers and controls are supported out the box, which are suitable for creating enterprise apps which do not have particularly taxing UI requirements.

Limitations of the approach

There are certain performance constraints which need to be understood: XAML views are parsed at runtime, which adds additional overhead when creating views – this is particularly felt in views with data templating (such as lists) which create many Xamarin views (in fact, one per item in a list).

There is, of course, a certain overhead involved in creating the native views (via the renderers). In most cases this is negligible but needs to be understood. Not all APIs are mapped, so some features may not be available, requiring the developer to extend renderers, or create new ones.

Developers need to be experienced and disciplined to get the most out of the platform, due to the patterns used. Junior developers could rapidly find themselves getting lost in a larger application with many renderers and services. The standard containers are limiting for creating non-standard/complex UIs. Performance of the container classes is pretty bad: view measurement invalidation is particularly over-zealous in Forms, leading to lots of lost performance.

How Xamarin perceives Xamarin Forms: "Xamarin Duplo"

This is currently a very hot topic on the Xamarin Forms forum. Forms has been a bigger success than Xamarin probably anticipated. In just one year, the Xamarin Forms forums have as much activity as any of the other forums which have been around for a few years. The latest release note had seven thousand views in 25 days.

Xamarin have been very clear about their intent for forms. Their website itself states that forms is for:

- “Data entry apps, Prototypes and proofs-of-concept,
- Apps that require little platform-specific functionality,
- Apps where code sharing is more important than custom UI,”

Apparently they internally named the project Duplo. The idea being, presumably that it would allow large chunky building blocks to enable developers to rapidly create simple cross-platform apps. This is a great strategy, aimed at the enterprise, and anyone who’s working in military, government, banking, or other major industries are carrying out due diligence for their respective organizations.

The simplistic/prototype app disclaimer is almost certainly an exercise in setting expectations. Xamarin are under-promising and over-delivering. Their documentation is littered with warnings to run for Xamarin.Android or Xamarin.iOS at the first hint of a performance issue. The forums and bug reports contain similar warnings from Xamarin staff in response to many issues users face.

UI / UX experience

Xamrin from Microsoft enables to write pretty much any kind of app you wish, using reusable Xamarin Forms code. We can even write complex view layouts and navigation using reusable code. Our current project has >99% code reuse (outside of our XF code library) across all platforms.

I’ve found my (appearing arrogant; but just being honest) vast and deep Adobe Flex knowledge to be shockingly applicable to Xamarin Forms development. Many of the tricks and techniques I used to employ when consulting clients on some of Flex’s toughest problems are transferable to Forms. I get the impression that this is similar to the experience of those coming from other disciplines, too.

The renderer system empowers us to create all manner of controls, and further more compose all kinds of behaviors. This gives us the ability to write small platform-specific “engines,” which do the heavy lifting for us, and then re-use them, headache free, throughout our Forms code. All the while churning out readable, MVVM compliant XAML, with testable View models, and a consistent familiar codebase.