

 **Telerik** A  **PROGRESS COMPANY**



AN ENTERPRISE ARCHITECT'S GUIDE TO MOBILITY

Best Practices for Enterprise Mobile Development Teams

CONTENTS

INTRODUCTION / 3

- 1. AGILE AND LEAN IN MOBILE DEVELOPMENT / 6**
- 2. THE IDEAL MOBILE DEVELOPMENT TEAM / 12**
- 3. DESIGNING FOR MOBILE FORM FACTORS / 19**
- 4. SEVEN KEY PRINCIPLES OF MOBILE DEVELOPMENT / 27**
- 5. ARTFUL TESTING WITH USER IN MIND / 36**
- 6. MOBILE DATA INTEGRATION OPTIONS / 45**
- 7. MANAGEMENT AND APPLICATION SECURITY / 55**
- 8. MAKING MOBILE ANALYTICS WORK / 60**

SUMMARY / 66



INTRODUCTION

Welcome to the multichannel world.

Odds are, you are reading these words on the glowing screen of a digital device. If you are like most readers in the developed world, you spend **55 percent of your waking hours in front of a screen**—or, more likely, several screens. The average information worker has three or more devices at hand, juggling business and personal communications. The makers of those devices shipped more than 10 billion of them in 2014 alone (roughly three billion more than there are humans on the planet).

The vast majority of those devices are mobile phones. In 2015, according to Gartner Group, sales of tablets will finally overtake those of PCs in all likelihood, permanently. The digital experience is now predominately a mobile experience.

For Enterprise Software Development Teams, Mobility is Now Priority One

Enterprise development teams that are just beginning to confront the mobile app challenge will face issues that smaller start-up organizations won't have. The enterprise team may be staffed, and have a management structure in place. They may have mature QA capacity and a process in place to deploy software through an experienced and organized IT Operations team. But mobile app development and deployment may present challenges for which neither the developers nor the operations staff will be truly prepared.

Desktop or web application developers who have adopted agile development methodologies may feel they have a good feel for rapid iteration and delivery. But mobile operating systems and apps are evolving much more rapidly than web technologies. Developers are likely to find that mobile is a very different discipline from web development.



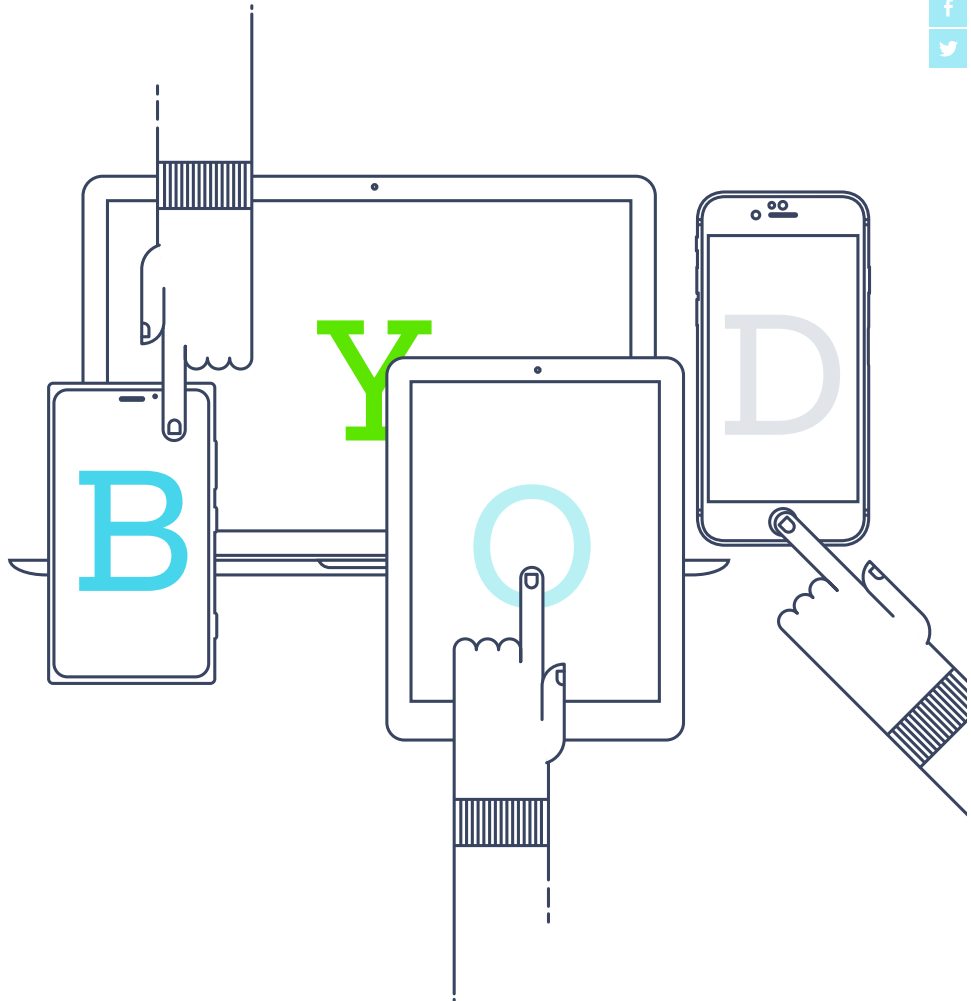


Thousands of Form Factors

An even greater challenge is the diversity of the end-user environments. When enterprise applications were exclusively browser-based and consumed via PCs, it was practical for an IT organization to limit the number of supported browsers and hardware configurations.

When an organization embraces mobility, those form factor constraints generally disappear. Users are commonly allowed to use their own personal devices, instead of or alongside enterprise-owned devices. With at least three major mobile operating systems, the mobile app developers must create user experiences that will satisfy end users of hundreds of different devices.

And user satisfaction takes on a new dimension in a mobile world. Enterprise web developers often saw it as sufficient to create functional but pedestrian applications for business users. A different standard has taken hold among mobile developers; **the current generation of mobile users expect a UX comparable to that which they get in consumer-oriented apps** they download from the Apple or Android app stores. That kind of commitment to UX design will be new to many enterprise developers.



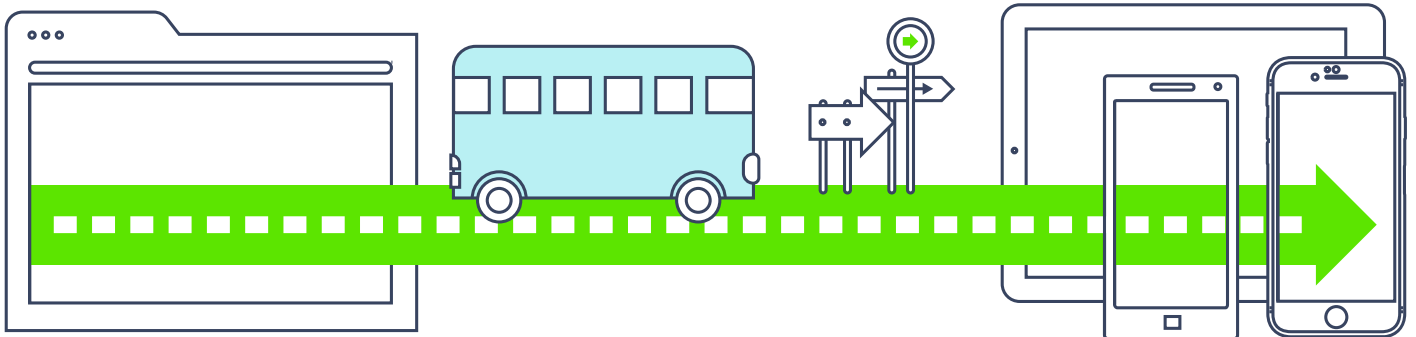
**An Enterprise Architect's Guide to Mobility**

is designed to provide a roadmap for enterprise development teams taking their first steps toward developing and implementing a mobility strategy—or for development managers re-evaluating their mobile commitments and looking for ways to be more effective.

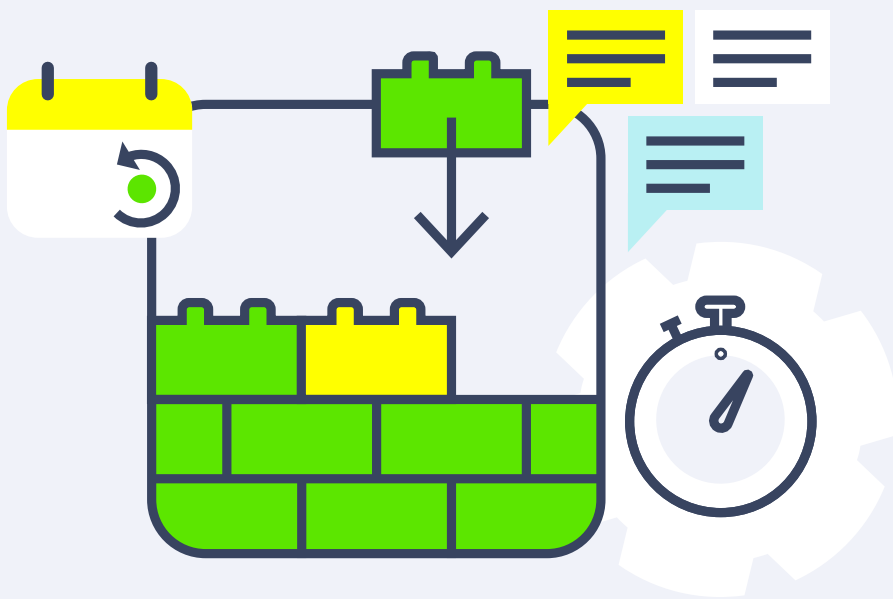
Telerik®: a Progress Company, a pioneering provider of tools and technologies to address the entire application development lifecycle, has gathered in this eBook a compendium of insight and expert guidance for the enterprise development leader on:

- Agile and Lean development approaches for mobile
- The ideal way to organize a mobile team
- Design for multiple mobile form factors
- Basic principles of mobile development (as distinct from desktop or web development)
- QA/Testing for mobile apps
- Mobile data integration
- Management and app security
- Mobile analytics

Our hope is to make it easier for enterprise teams to smoothly transition from web to mobile development by sharing some of the most important experiences and best practices that Telerik has collected in creating tools for this evolution. This is just one more tool to support the mission Telerik has had since its founding in 2002: to enable the developer community to create one-of-a-kind app experiences for their customers.



1



AGILE AND LEAN IN MOBILE DEVELOPMENT



AGILE AND LEAN IN MOBILE DEVELOPMENT

Since the 1970s, enterprise software developers have been debating the merits of traditional, “Waterfall” development methodology versus incremental, iterative development methods now collectively known as “agile software development.” These are virtually polar opposite philosophies. The Waterfall approach treats development as a strict sequential process, from the application’s conception, through end-to-end design, to implementation, verification and maintenance. Agile methodologies divide the development process into a series of iterations in which the application is designed, built and verified by the business owner in small increments, each functionally complete, and gradually accumulating the required functionalities of the finished application.

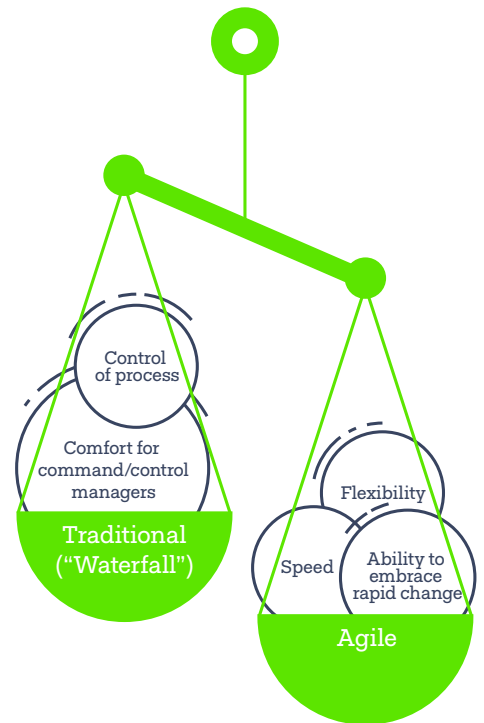
Developers of desktop or web applications may continue this debate for years to come. But mobile development is another story. Mobile is in an early stage of its life cycle and is evolving rapidly. **Mobile, by its nature, belongs to the agile developers.**

Embracing Uncertainty

An enterprise establishing its mobility strategy is moving into an unknown territory, where software development practices are immature compared to the desktop or web development world. So it makes sense to adopt a set of principles that **embraces uncertainty**, as opposed to trying to drive all the uncertainty out of the process up front.

Uncertainty arises for a variety of reasons in mobile, but mostly because so much of the environment into which mobile apps are launched is outside the control of the enterprise and its development team.

Many enterprises now building mobile development capabilities also are in the process of embracing the **Bring Your Own Device (BYOD)** trend, in which employees are allowed to use their personal mobile devices to run corporate applications and access corporate data.





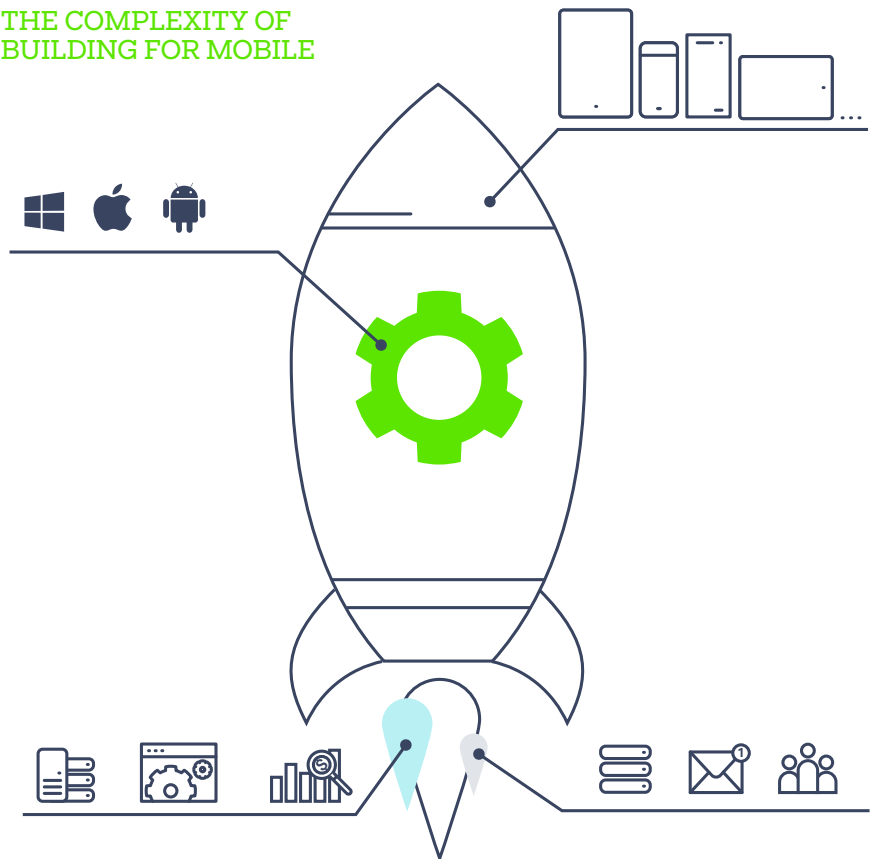
Thousands of Variants

With that, the enterprise accepts responsibility for supporting hundreds of distinct devices running at least two different operating systems (Apple's iOS, Google's Android and possibly Microsoft's Mobile Windows OS as well). With all the form factors that must be considered for these devices, the app developer may need to allow for thousands of new platform variants.

Also important are operating system dependencies that are unfamiliar and that change rapidly. At any point in the development cycle, the team could get an iOS update from Apple. The team may know in advance that it's coming, but it still sidetracks the iOS version of a new app, and the team must scramble to deal with that at the eleventh hour.

Those things don't happen regularly, but at mobile's current state of maturity, it is a more common occurrence than in desktop development.

THE COMPLEXITY OF BUILDING FOR MOBILE





A Different Paradigm

Imagine your team is tasked with “mobilizing” an old line of business system that you’ve had on the desktop for 15 years. It’s not as simple as saying, “Let’s take this order or PO screen and display it on an iPad.” The screen in the desktop app may have 75 fields; it simply will not work to pull all of that information into one giant scrolling mobile form.

Mobile is not just a new device platform; it’s a **different paradigm**.

You don’t know what your end users are going to want at the outset. So you iteratively develop a prototype and put it in front of the customers very quickly, so they can play with it and comment on it. Only do then you start building... a little at a time.

That, in a nutshell, is **agile development**.

The term “agile” means a number of things. Spelled with a lower case “a,” agile is set of principles: Planning only what is necessary for each iteration; not speculating on design; encouraging tight feedback loops over the life cycle of the project. Capital “A” Agile is a set of institutionalized processes. There are several documented variants, the best known of which are Scrum or **Lean Software Development**.

Agile methodologies promote the ideals of rapid adaptation, evolutionary development, rapid delivery, continuous improvement and flexibility in the face of change. These are essential qualities in a mobile development team.

Agile teams meet frequently—there are daily stand-up meetings, and each “sprint” (typically a two-week design and development cycle focused on a subset of the application) will culminate in a review session involving the product owner from the business side.

In general, the team gets better at measuring its progress and delivering value. Based on what the team estimates in advance and what it actually delivers over several sprints, they come to know their velocity.

It requires commitment to all of the practices, including the retrospective, in which the team looks back after two weeks and asks, “What did we not do well? Why were we off on our estimates of this task? Why did we miss half of what we had in our backlog for this sprint?”



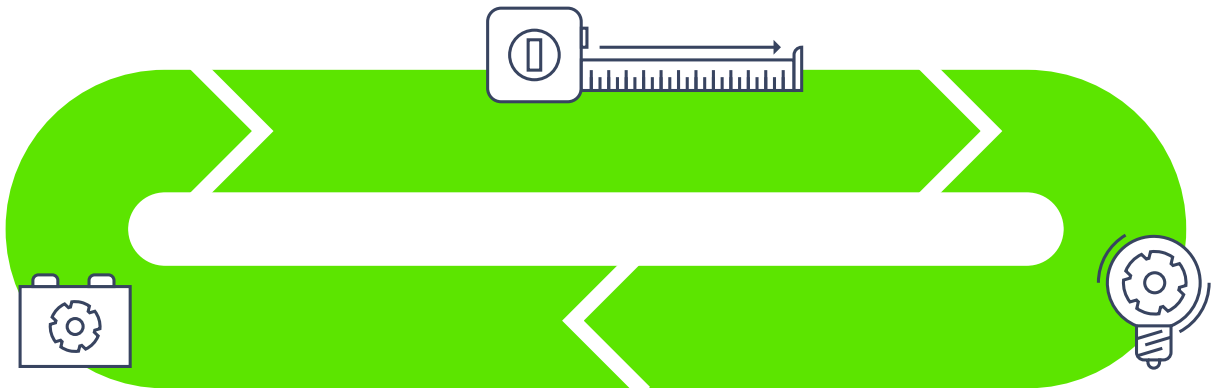


Lean Software Development

The Lean methodology grew out of the pioneering production system at Toyota, and was gradually applied to software development in the early 2000s. Lean is another way of aggressively minimizing waste, having tight feedback loops and other principles of agile development. But Lean is not as prescriptive in terms of what the developer needs to do to be successful as are some of the more formal Agile methods.

A Lean development team typically will evolve something akin to a product management function, with Project Managers and User Experience designers responsible for talking to customers, building out plans and roadmaps, building prototypes and iterating over prototypes with engineering and with customers.

Adopting Lean means engaging customers as early and often as possible, and shipping software as often as possible. Part of engaging customers is showing them prototypes—rough ideas of what the team is doing. It means defining **Minimum Viable Products**—shipping software the customer can use, even if the products of each iteration are not visible to everyone.



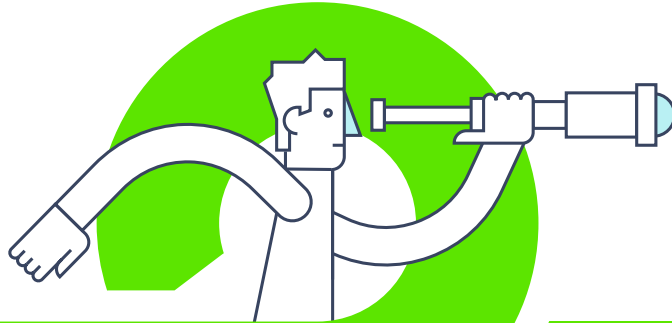


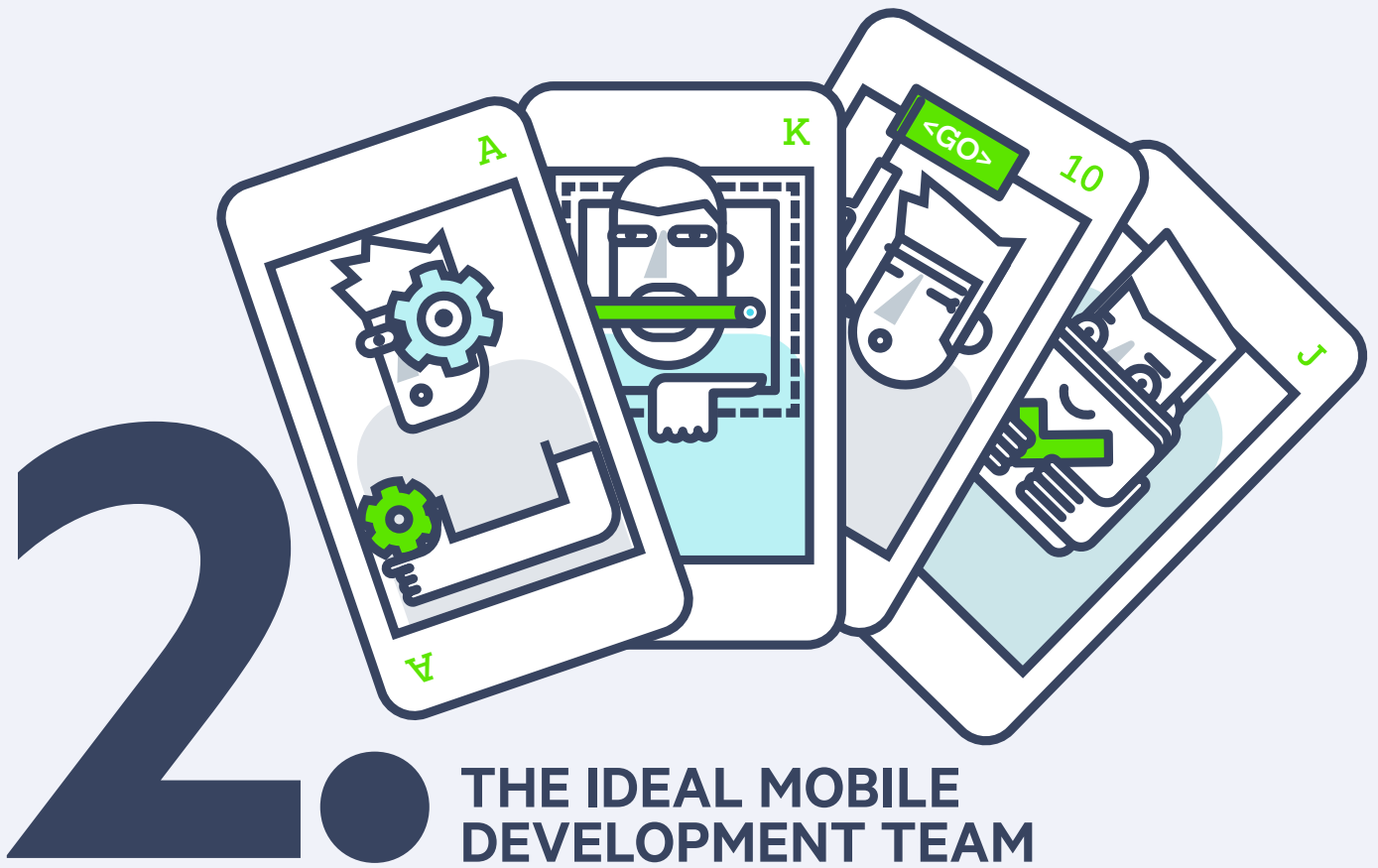
How Much Involvement from the Business?

In formal Scrum, sprints run two weeks. So, a common question is: in an enterprise setting, is it practical to ask for involvement from the business every two weeks?

The business doesn't have to be willing to commit everybody at the end of each iteration, but it should be willing to commit one person. That individual—in agile parlance, the **Product Owner**—has the proxy for the business in evaluating successive releases against the requirements defined by the business.

Agile development methods succeed in executing the enterprise mobility strategy when the **developers and their counterparts on the business side recognize that they are in this together**. They have common objectives, and they share the challenge of embracing the uncertainties of mobile development as partners.







THE IDEAL MOBILE DEVELOPMENT TEAM

An enterprise that is establishing a mobile development team may transition experienced developers from its existing team, or bring on new engineers with mobile-specific backgrounds. Either way, the mobility team will have to be organized for flexibility, familiarity with the mobile paradigm and close collaboration with the Product Owner from the business side.

Who Should Be on the Enterprise Mobile Team?

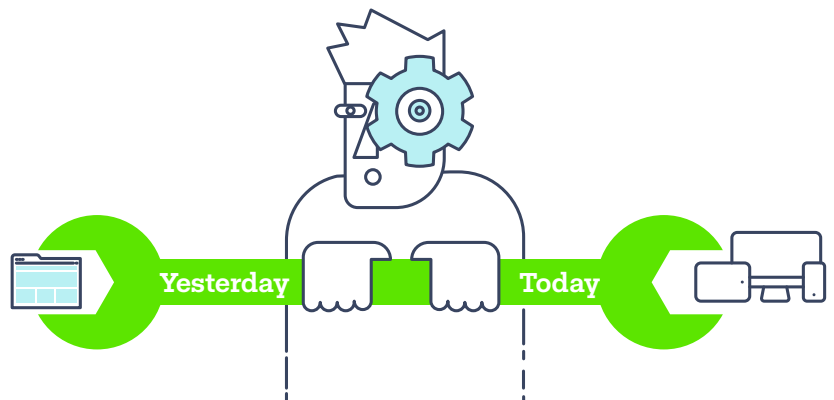
In transitioning developers to the mobility team, focus on current web developers, as opposed to the desktop team. The web developers will have built up their skills through .NET or Java or PHP or Ruby On Rails. **The common denominator is that they were modernizing legacy systems via the web.**

The team is likely, at least for a time, to adopt **hybrid development**—an approach that combines conventional web skills with mobile development.

[More on hybrid development in Chapter 4.](#) 

Hybrid development makes sense from a technical perspective when “mobilizing” applications originally developed for the web, especially as the finished app is likely to retain the web interface for users who need to access the back end data and application services through a browser. Hybrid also is an effective way for developers transitioning from the web team to keep one foot in their comfort zone.

In most enterprises, projects are led by the project manager/team lead. Sometimes that will be the senior developer, although development experience is not always effective preparation for leadership. It may be that the best team leads will not be senior developers, but rather a career manager whose real strength is in bridging the communication gap between engineering and the end users who are defining requirements around the strategic needs of the business.



How Centralized/Decentralized Should It Be?

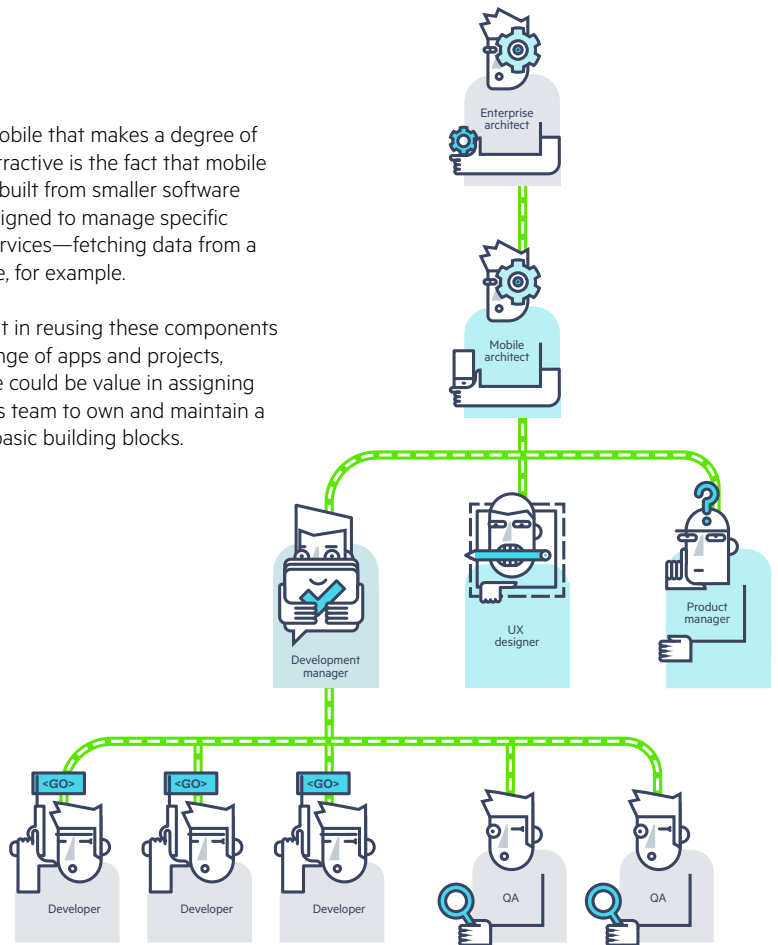
Many enterprise technical organizations have shared services supporting the project teams. There is a benefit in consistency within an enterprise. That is why enterprises have internal style guides for written communications, and why corporate development teams insist on using common code components and support only a limited set of tools.

This does not necessarily imply a centralized app development team. At Telerik, all of the UX designers report to one manager, separate from engineering, to promote consistency of UX across all platforms.

Centralizing development gives IT a degree of control over the product roadmap and strategy, freeing the team from the need to support multiple vendors' offerings that may not be compatible and could lead to poor coding choices. This sort of control is important in desktop or web development; it is **essential** in mobile development, given the accelerated pace of change.

One aspect of mobile that makes a degree of centralization attractive is the fact that mobile apps tend to be built from smaller software components designed to manage specific functions and services—fetching data from a specific database, for example.

There is a benefit in reusing these components across a wide range of apps and projects, suggesting there could be value in assigning a shared services team to own and maintain a library of these basic building blocks.



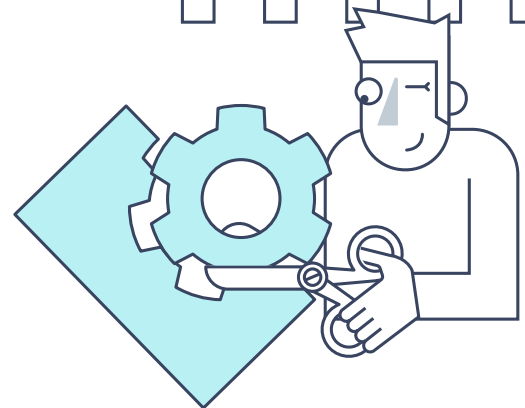
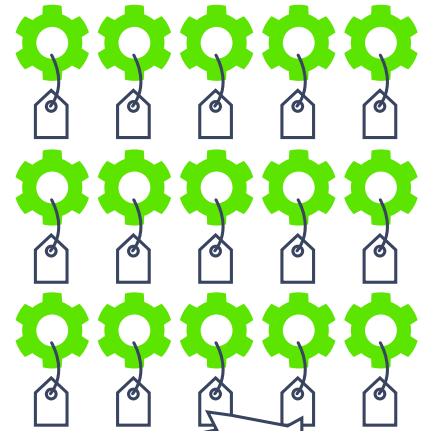


Build or Buy?

An enterprise may have a development team that actually builds these components, and shares them via a code repository for other development teams to use. But there also are opportunities to populate that repository with widgets from commercial vendors, or for that matter from the open source world if the CIO is comfortable sourcing that way. If apps need a common calendar widget, the project team can specify the back-end data but does not have to build the widget itself.

On the other hand, you want to avoid centrally controlling resources so tightly that teams can't get what they need when they need UX help. If you have five designers and 25 products, people will be fighting scarce resources. Some of this depends on how federated or centralized the organization's own business model is.

If the enterprise tends to maintain all of its legacy business systems centrally, then there is benefit in a central team. But if it is more of a federated organization made up of wholly owned subsidiaries, there may not be as much need for centralized control of development or uniformity of UX, because the business units have more autonomy. If the business feels overly constrained, that can be a problem for adoption.



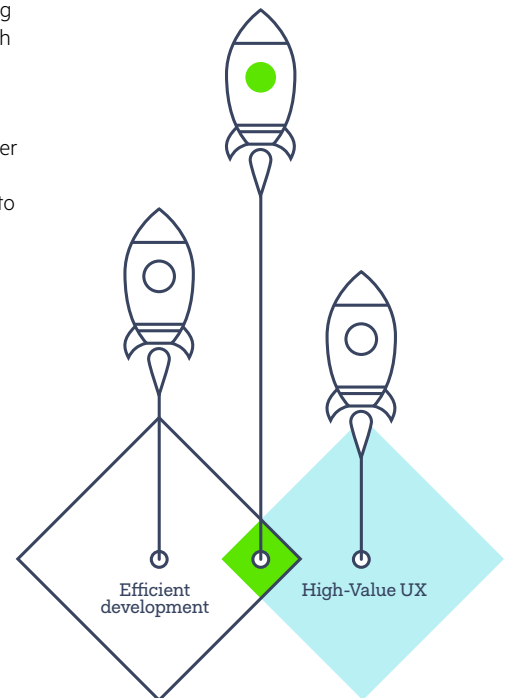


What is the Role of UX?

An engineering background doesn't always prepare the developer to see the innate value of **User Experience (UX)**. As we will see in Chapter 4, end users of mobile apps have much higher expectations for the quality of the UX than was typical in the enterprise desktop or mobile world.

Many developers—especially in enterprises—are accustomed to developing “battleship gray” desktop and web applications. But when you have a manager who understands the value both of efficient development and a high-value UX, it is more likely that the process will allow UX to influence design, encouraging iteration and collaboration between back-end developers and UX designers. A great developer might not have the same appreciation for the business issue being addressed by the app, and might be more concerned with the development schedule and with the performance of the code, with less regard for what is coded is on the roadmap, or how it got there.

Optimally, product management and UX live outside of engineering and can take a very UX-first approach. The ideal approach is to look at the customer problem that the business is trying to solve and design for that. When you start with the feature in mind, you're solving an unstated problem. You may even be solving a problem that the business doesn't have. Development teams that focus on the problem as the customer sees it are likely to waste much less time and resources iterating over things that don't need to be developed.





Still, even in mobile, you want your sales force automation app to have a similar UX to your HR app, because you have the same people using both. There is a cost to the user having to use multiple apps if there is cognitive friction between how you execute a very simple function in one versus the other. But if apps are developed from a common enterprise toolbox of functional components, they are likely to inherit consistency.

UX oversight is not just a question of design, but also oversight of functional implementation—of how that function is coded and what components are used. The ideal is to establish a “design umbrella”—a team whose skills include those of a front-end web designer, an interaction designer, a usability designer and an information architect. Sometimes one person will contribute several of those skills.

What's the Role of QA/What Should Their Background Be?

It is increasingly common to include QA in the skill set of the project team. QA people may not be developers themselves, but they do require some level of development knowledge, and in many enterprises, developers do have the responsibility for QA.

The key is not that the QA person has lesser skills or different training. The key effective QA can only be achieved by someone who didn't write the code being tested. If the code is your own, in the back of your mind, you don't want to break it. It is important to delegate QA to someone who is paid to break the code. That involves a distinct, highly valued skill. The tester has to be very deliberate about finding errors that users will discover by accident. That is not an easy thing to do consistently.





Whose Responsibility Should It Be to Track Mobile Analytics Data?

Analytics—crash reports, KPIs, dashboards—are the key metrics for app performance and integrity. They may be of interest to QA, but they are really designed to enable developers to monitor their apps. Other consumers of these analytics may be team leads, product managers (if that function exists within the enterprise) and the service desk supporting the apps.

[More on Analytics in Chapter 8.](#) 



The Product Manager's Role In Enterprise Mobile Teams


Product management isn't traditionally an enterprise role, but this is beginning to change in organizations committing to mobility, as enterprise mobile apps tend to use many of the design and UX conventions seen in consumer apps, and mobility teams increasingly think of end users as "customers."

In agile development, there is a designated product owner who represents the voice of the customer. This is more critical in mobile than it has ever been before, because the developers need to be iterating faster than ever.

It is critical to have an individual responsible for articulating the customer's needs—in agile terminology, the "Single Wringable Neck." The product management role more typically is a developer team lead function—the provider of the app to the business customer—but the business may want to maintain this responsibility and delegate it to the Product Owner.



3



DESIGNING FOR MOBILE
FORM FACTORS



DESIGNING FOR MOBILE FORM FACTORS

Enterprises that embrace the Bring Your Own Device (BYOD) trend have accepted the challenge that most sharply differentiates mobile development from other disciplines: the diversity of screen configurations in which an app must provide a satisfying user experience.

Start with device size and screen orientation. A mobile app has two orientations—portrait and landscape, depending on how the user holds the device. In addition, there are multiple types of mobile devices with different screen sizes and resolutions—tablets, phones, phablets and so on.

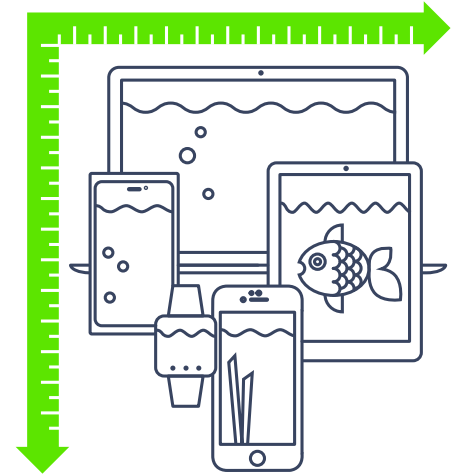
Because users across an enterprise may adopt devices of many vendors and models, the combinations can grow into the thousands.

Samsung alone, across its mobile product line, has more than 520 screens.

Now, there is a huge push for wearables. Many things will change when you get to screens small enough to be practical for wearables—menus and similar presentation conventions on the screen, and even the physical characteristics of the device such as buttons and other physical, tactile controls—will vary widely among devices.

Differences Start with the OS

Of course, there are significant differences between the operating systems, in features like tab strip menus versus sliding “drawer” menus. The OS will decide for you where such features will be placed. If you design an app that has the tab strip at the bottom and run this on Android, the OS will place the tab strip at the top, because Android users are accustomed to seeing the tab strip at the top of the screen.





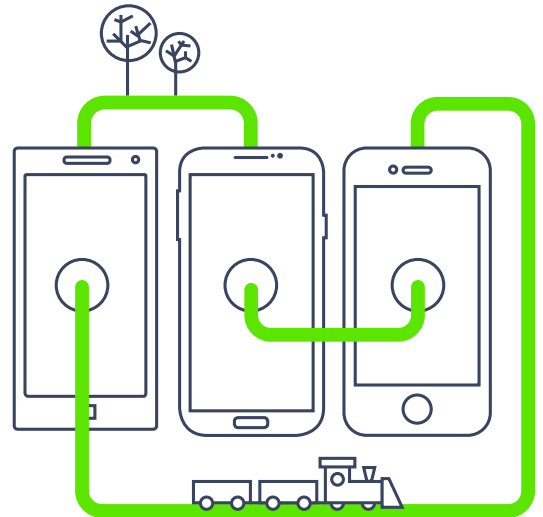
The enterprise needs to have a good idea of the range of devices on which an app will run, and look for appropriate approaches as opposed to allowing QA to discover issues with certain devices and configurations retrospectively, as issues to be corrected later.

Organizations using **prebuilt software components** that provide specific functions and services will find that these widgets are platform-specific, so they can perform natively and provide a native iOS, Android or Windows experience. But in terms of the developer's experience, the components are part of a broader **framework**.

If the project is an Android app, the project team will need to choose widgets that leverage Android native script, but once that choice is made, the framework will enable the developer to use those components without worrying about the OS specifics. The framework encapsulates everything a developer would need to know about Android in a specific abstraction layer. If you need a button, you simply specify a button, and it will apply the Android button.

An Exception for iOS

The framework is a development environment, which may be local on a desktop computer or run as a cloud service. One important reason a team might choose a cloud service is that it is a PC-oriented enterprise that needs to develop an iOS app, because iOS apps generally can be built locally only on a Mac. There are vendors who provide the framework in the cloud specifically to enable non-Mac development teams to build iOS apps, using the vendor's Mac servers.



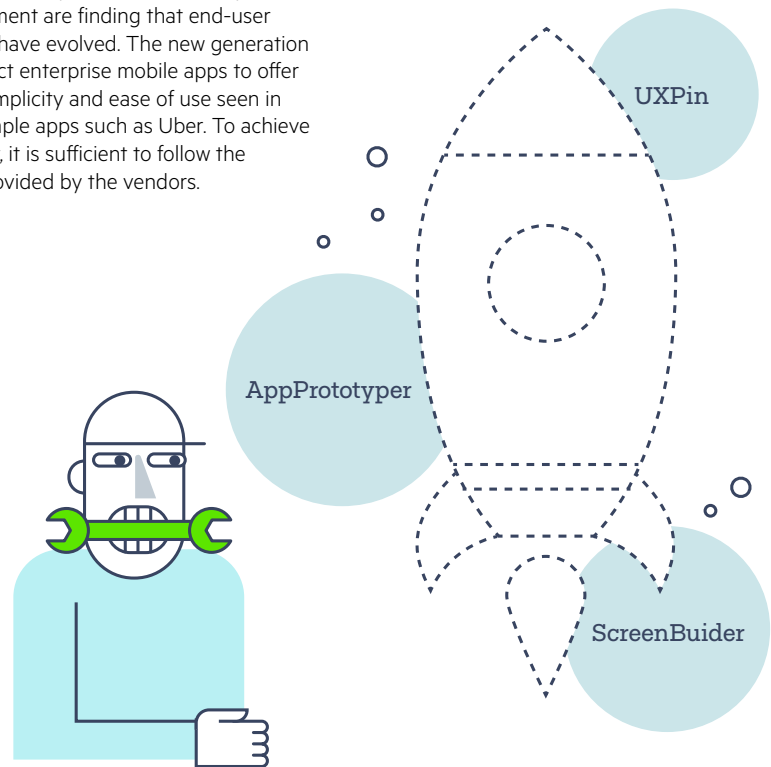


The iOS instance is exceptional—generally, the build process is local. However, another phase where developers are likely to work in the cloud is in **prototyping**. Prototyping tools allow the team to design mock-up screens for development; tools like Telerik UXPin or AppPrototyper enable the coder to design for one form factor, and automatically create design templates for other form factors. Those tools can be accessed from the cloud.

The prototyping tools do not actually generate code, just static images useful during the design phases. There are tools that can generate actual code, such as the newly introduced Telerik ScreenBuilder—a tool that actually enables drag-and-drop manipulation of code entities, not just static images. The developer can position UI components, bind lists to specific data collections and so on. This process will generate what is called “**scaffolding code**,” which may be thought of as the next generation in prototyping. (Scaffolding enables automation of a substantial portion of the development process, but projects will generally perform additional coding to complete the app.)

Complying with Apple's and Google's Design Guidelines

As we saw in Chapter 2, developers coming to mobile from their experiences in desktop or web development are finding that end-user expectations have evolved. The new generation of users expect enterprise mobile apps to offer the kind of simplicity and ease of use seen in consumer simple apps such as Uber. To achieve this, generally, it is sufficient to follow the guidelines provided by the vendors.





Apple, Google and Microsoft publish lengthy, detailed and highly prescriptive style guides for their platforms, with conventions for layout and functionality. Vendors of development frameworks for these OS environments document their own standards, as well. It is useful to know this documentation exists, particularly for enterprises developing in NativeScript. But the use of pre-built software components and development frameworks makes it largely unnecessary to consult the guidelines—the tools essentially apply these standards for the developers.

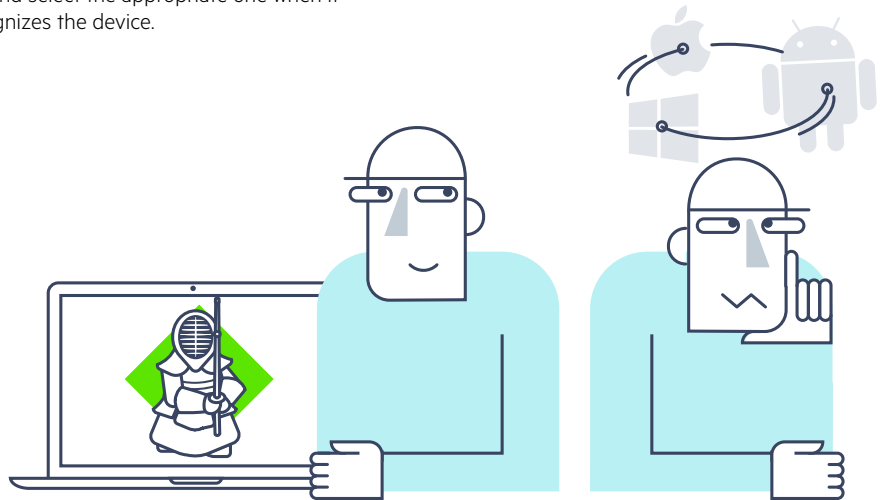
Leveraging Pre-Built UI Components

Many mobile UI libraries (such as Telerik Kendo UI® framework) combine all those design styles and other conventions needed by developers.

If you build a simple application that has two tabs on the bottom, reflecting the style for iOS, it will look like a native app. If you run the same app on an Android device, the framework will automatically move the tabs to the top, in accordance with a different style sheet for Android. The product offers this same kind of abstraction for Windows phones, as well.

Selecting a Mobile Development Approach

There is an important difference between building UI for a mobile website and building it for a mobile app. For a site, it may be feasible to engineer a single UI that recognizes the platform—phone or tablet—on which it is displaying and adjusts itself to that environment. In practice, it is very difficult to accomplish this with an app, which has a wider variety of controls optimized for one environment or the other. A universal app will have two distinct UIs and select the appropriate one when it recognizes the device.

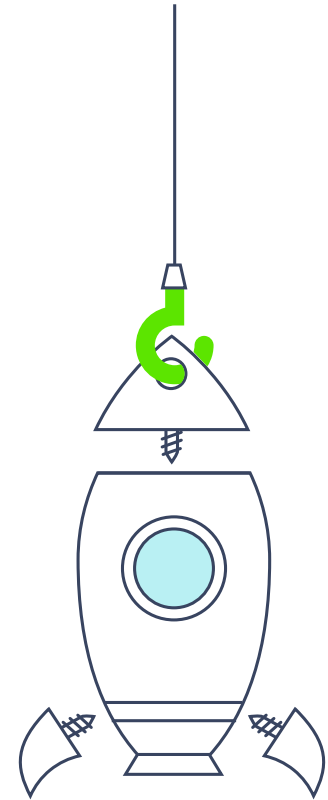




The ideal is to build responsive **“universal” apps** that adjust their user interface to the device on which the user is running them—phone-specific UI for a given phone, tablet UI for a tablet, with effortless adjustment for the orientation of the screen and instant recognition of the OEM customizations that a device manufacturer may have made for the specific model and OS.

A range of approaches can be supported using available frameworks. The choice depends on the app scenario. Each provides UI components, as well as design templates, application building blocks, image editing capabilities, and integration with backend services.

- Mobile web is the least expensive approach and has the best reach (covers the widest range of devices), but might not fit scenarios in which you need to use specific device capabilities or do advanced transitions.
- HTML5 is the preferred approach for responsive apps, but performance is limited by the capacity of the devices (although this is improving continuously).
- Native is an expensive approach, but skilled native code can sharply enhance performance in a specific OS. Native will require the enterprise to find programmers with native skills (or outsource native coding to vendors who specialize in it) and maintain multiple code bases. It is appropriate for the most polished apps, especially those that will be used by consumers and whose UX will reflect on the image of the enterprise.
- Hybrid, a web app wrapped in a native container, offers both ease of development and a native-like look and feel, as well as the ability to use OEM device capabilities.





Testing and Distribution

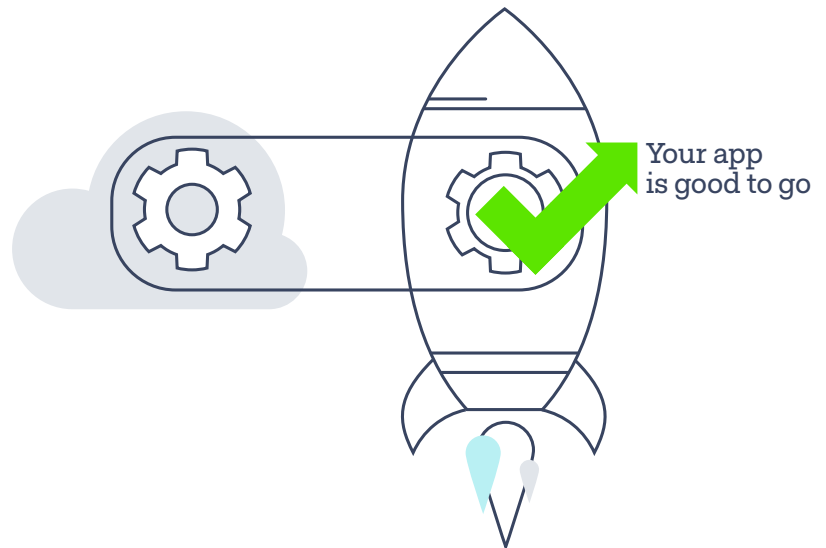
Testing is one of the most challenging issues in supporting the many devices and form factors. Testing for performance and functionality on multiple form factors is done generally in the cloud. Vendors like BitBar have established “device cloud” services that allow the developer to rent physical devices by the minute, for testing. They maintain a warehouse of devices and will send screen shots to illustrate how an app looks on that specific device, in different screen orientations.

An enterprise will require an easy way to distribute apps. MobileIron and Airwatch have enterprise mobility management (EMM) solutions that allow you to distribute apps to a specific segment of your employees.

New functionality in this space includes an easier way to report bugs and functionality issues from mobile devices, or from the app. Telerik has a function for this: if you shake the phone, it takes a screenshot of the app, and you can write on the image to describe specific problems. Then with a simple point and click, the report is sent to the developers.

How Can Lean Methods Be Applied to Mobile Development?

The granularity of mobile apps keeps development simple for enterprises and lends itself well to agile development. There is less QA time and easier deployment—updates and new apps can be obtained remotely, often without the user’s needing to intervene or request it—and deployment is more frequent and faster.

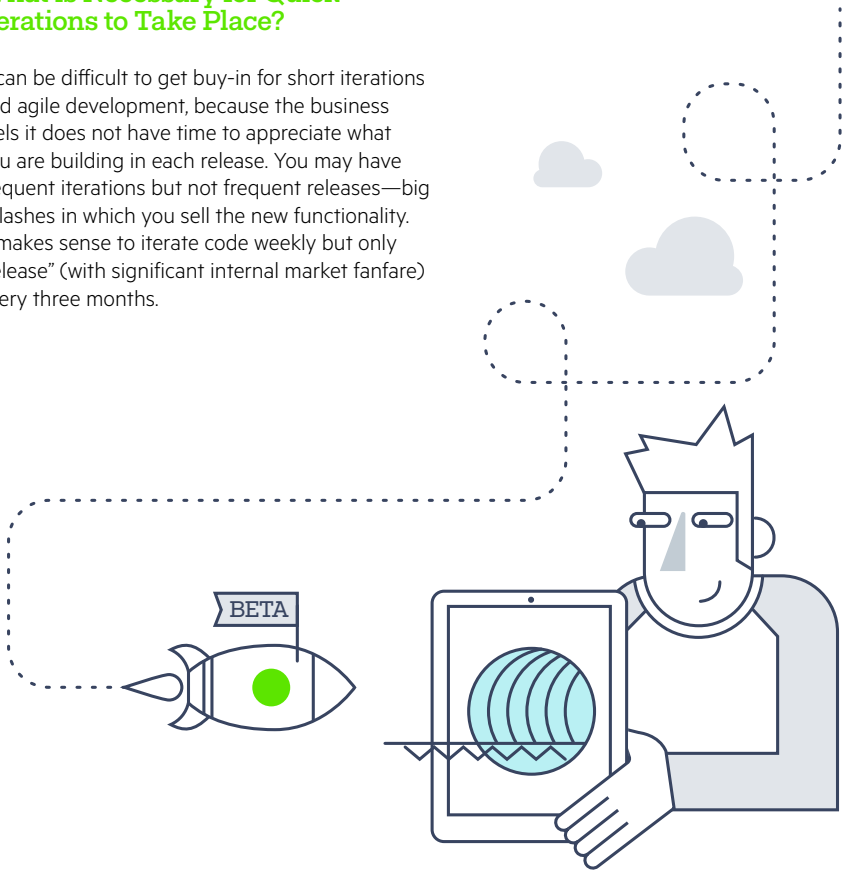




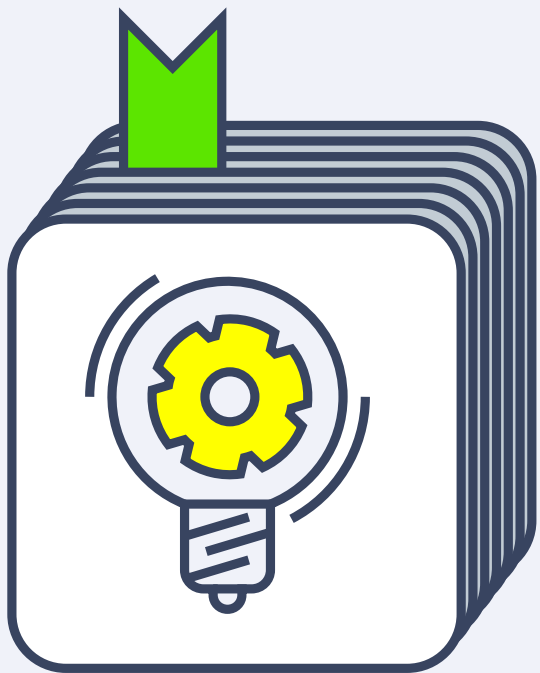
Agile development for multiple form factors leverages the prototyping we described earlier. You can introduce a range of design ideas to prospective end users before you write a single line of code. That allows you to discard the ideas about which your users are not excited before they're even realized as features. This approach can help enterprise developers become stronger internal marketers, as well as coders.

What is Necessary for Quick Iterations to Take Place?

It can be difficult to get buy-in for short iterations and agile development, because the business feels it does not have time to appreciate what you are building in each release. You may have frequent iterations but not frequent releases—big splashes in which you sell the new functionality. It makes sense to iterate code weekly but only “release” (with significant internal market fanfare) every three months.



4



SEVEN KEY PRINCIPLES OF MOBILE DEVELOPMENT



SEVEN KEY PRINCIPLES OF MOBILE DEVELOPMENT

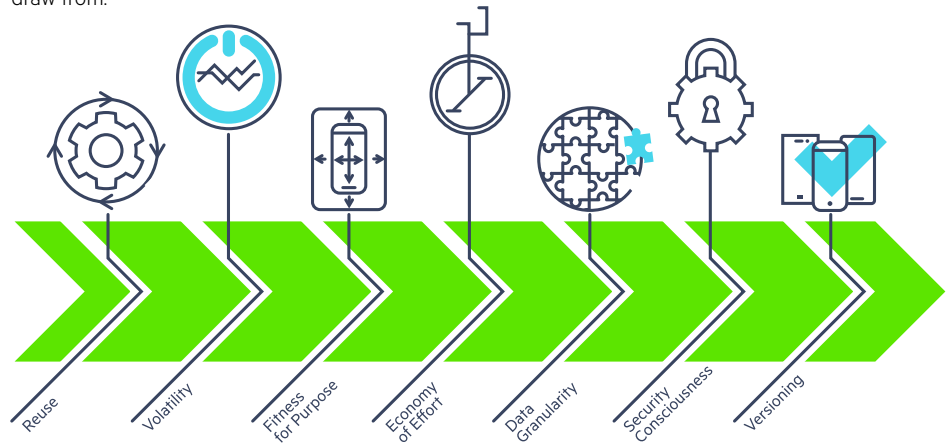
A decade ago, desktop development moved toward Service Oriented Architectures (SOA) and it became common for applications to rely on shared services that lived in a central enterprise hub. The development team might find five different internal apps that all shared the need to check inventory, and which, formerly, were housed in the mainframe. A typical project would be to modernize those applications by putting a reusable front end on the shared service of accessing inventory data, so that any of those apps could call that service in a uniform way. There typically was a Shared Services team that was responsible for exposing this service, building out the infrastructure and building up those endpoints, versioning and the like.

Mobile development is similar but abstracted to a higher degree. Instead of an inventory service, you have an inventory widget that connects to that backend data and provides the view that you need to use, in one mobile app for the sales force and another app for marketing. Components now find their way into the UI.

What are the key coding principles/standards the developers should adhere to?

1. Reuse

As a developer, you have access to a much better and broader set of tools, both from commercial vendors and from the open source community. You have a library of patterns that you can draw from.





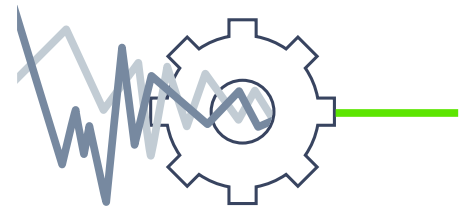
Fundamentally, code is still code. Up to a certain level of complexity, the mobile developer uses the same skills and talents that a web or desktop developer uses. What mobile has taken farther than the other disciplines is reuse—the practice, and the tools to polish and leverage the code to provide distinct functions and services in the form of reusable components.

2. Volatility

An important example is the volatility of network access. In an enterprise, the development team may be close to the data center that houses the backend data accessed through the app. But the user may have strong connectivity one moment, and be in a tunnel the next.

Effective mobile development requires **a defensive approach to maintaining performance despite network connectivity that may be intermittent.** The ability to use an app either online or offline is fundamental to mobile app design, whereas it has not been as critical in web design.

This addresses a basic user expectation. With a web application, the user doesn't assume that a blank screen is the application's fault. It could be a network connectivity or browser issue. By contrast, a mobile user is much more likely to blame the app in such situations, because mobile users are used to apps that handle network dropouts gracefully.





3. Fitness for Purpose

There are developers who make the transition from web to mobile with their code integrity and UX standards intact, and those who don't. On a 3-inch screen, a 7-inch screen or a 9-inch screen, your app can look good and satisfy the business requirement in an elegant way—or not. The difference is not necessarily in the successful adoption of new coding techniques—it's in **the developers' commitment to ask themselves, "What are you building, and why are you building it?"**

A successful developer will make the UX transition as well as the coding transition. It's an application, like an application on any platform, but how do you make it light up a mobile screen?

There's a difference between a consumer-oriented app—one that is intended to be distributed to customers through a public app store—and an app intended for the employees of an enterprise. A consumer app can do a lot to increase a company's visibility and share of mind, and that can have a direct impact on revenue. But it's fair to ask, does the application, which started on the web as a browser-based tool, really need to provide a mobile experience? Does that add to functionality?

You can ask the same question about an employee app, but the answer is much more likely to be "yes." Effective apps are built because the phone or the tablet adds some native platform capability that isn't part of the web platform, or because mobility itself is of practical value.



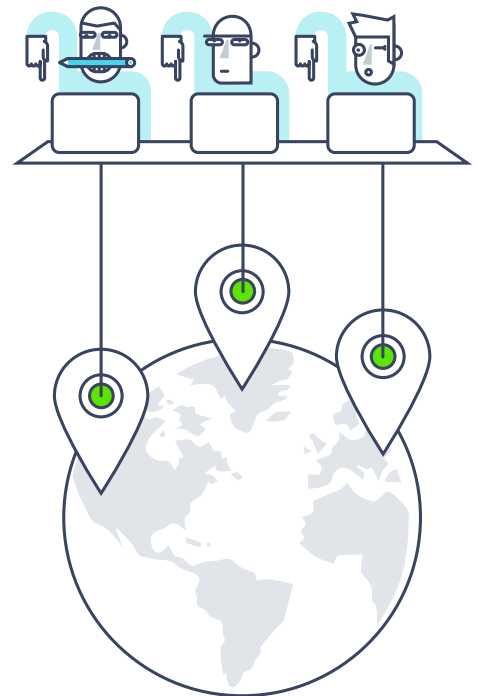


Take the example of an insurance application, which may be built around the heavily advertised capability of paying the premium on a smart phone. The practical, differentiating value to the consumer for such a function is debatable. However, the additional concept of taking and transmitting a picture of the damage from an accident genuinely uses a native mobile feature, and is a much more obvious example of value added through mobility.

You use geolocation, or you use fingerprint identification to authenticate; you use the device's camera to get high resolution images. **When you start to think of what a mobile device can bring to the context of the business problem, it changes the way you think about the device itself and the code itself.** The objective is not to duplicate a desktop application on the phone, learn a new language and replicate the full range of desktop functionality on one device, let alone five.

4. Economy of Effort

Enterprise developers have, for years, been under pressure to do more with less, and have seen significant reductions in their headcounts. A decade ago, a large enterprise might have 300 developers assigned to a major e-commerce project for the web, a project that might have a multimillion dollar budget. Today, budgets and staffing are reduced, and a portion of the work typically is outsourced. Outsourcing adds an additional layer of complexity to the project and development discipline. The outsourcer's staff engineers are farther removed from the business objectives than the staff engineers are. They live or die by the wireframe.



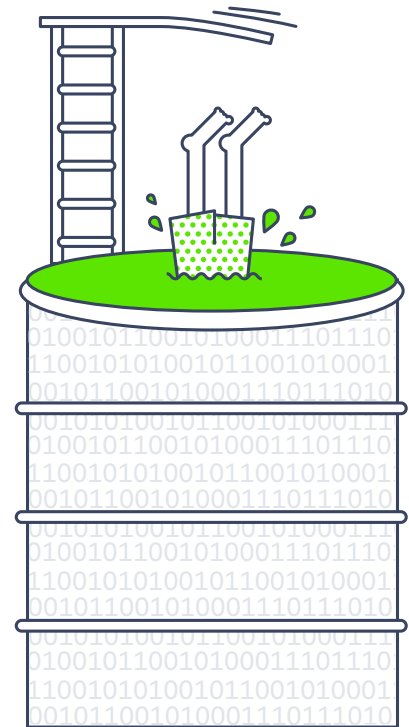


Developers are going to be most productive when they're using the skills and frameworks that they already know, that they've already mastered. The so-called **hybrid development** approach hits that sweet spot, because the developers are working from their existing base in web application development—HTML, Javascript, CSS—to develop mobile apps.

Hybrid is the intersection between the use of regular web skills, development for a website, and wrapping that code in a native container (via the Cordova platform). When you do that, you can install the code natively on mobile devices, so that the app can access native device features, such as contacts or geolocation data. Hybrid is ideal for targeting multiple mobile platforms, as would be required in a BYOD scenario. It is a minor compromise; hybrid apps generally don't provide the advanced graphics performance that you can only get from a native mobile app.

5. Data Granularity

One place where mobile developers can draw inspiration is from the gaming field, where developers have become highly adept at downloading updates mid-game, based on changes on the server, effectively scaling what needs to work on a low-powered device. Most enterprise developers seem to forget that these are ARM-based devices that cannot process 100,000 list items all at once, whereas in a desktop environment, the user's machine is very likely to have a stable network connection and be capable of processing much larger amounts of data. In a 4G world, five Gb is an enormous amount of data to send down to a phone.





A mobile developer, in short, must code with the assumption that the network will drop out at frequent intervals. The app has to deliver data to the device in smaller increments, to give the user a good experience.

6. Security Consciousness

Web developers are accustomed to hiding their APIs behind firewalls. Their only container to the world is a web browser; they poke through the firewall behind the DMZ into an application server and have free rein to do essentially anything they want to those APIs, because they know the connection between the servers is secure.

In the mobile environment, however, the **APIs are exposed to the world**, and the developer must be more security conscious. The developer must programmatically lock down the API, rather than working with it inside a secure site.

Mobile developers clearly need to be more security conscious, and must address the protection of the data that is sent to the device in anticipation of the network dropping out, how it is cleared out, whether it is encrypted and so on. And, the user is not only consuming data, but entering it. So the app must be able to manage **frequent state changes**—network-connected or not, high or low bandwidth—holding data entered offline until the network connection is re-established and then transmitting it, in a fashion that feels seamless to the user.



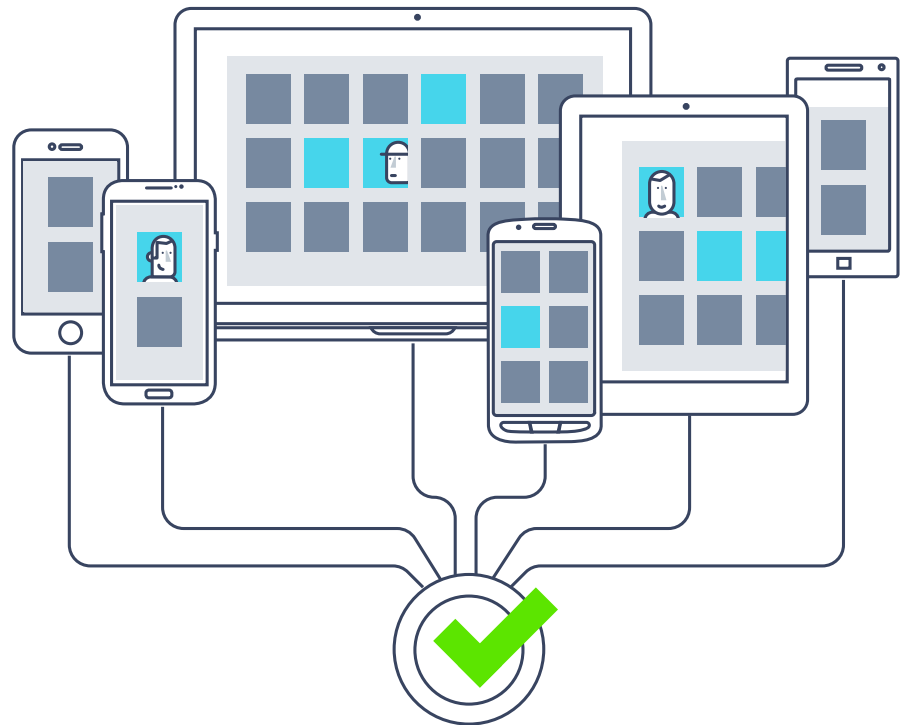


7. Versioning

Another important type of volatility is in versioning of mobile platforms. In a BYOD organization, the app store may contain multiple versions of the app—not just for different operating systems and device manufacturers, but sequentially different versions of the app for each given OS and device generation, since users typically are on their own in terms of updates.

If you deploy through a public app store like iTunes, it may take weeks for the new release to be reviewed and deployed, and once it does appear, whether and when the users update it is beyond the control of the developers. Even a managed deployment can take several days or a week.

Many enterprises do distribute apps through the public stores, as a matter of convenience. If they want to take advantage of more device-specific or enterprise-specific capabilities—security policies or management policies—then **enterprises may use enterprise mobility management (EMM) services**, which are cloud-based. Even then, they may choose to distribute apps through a public store, but also have their own private app store.





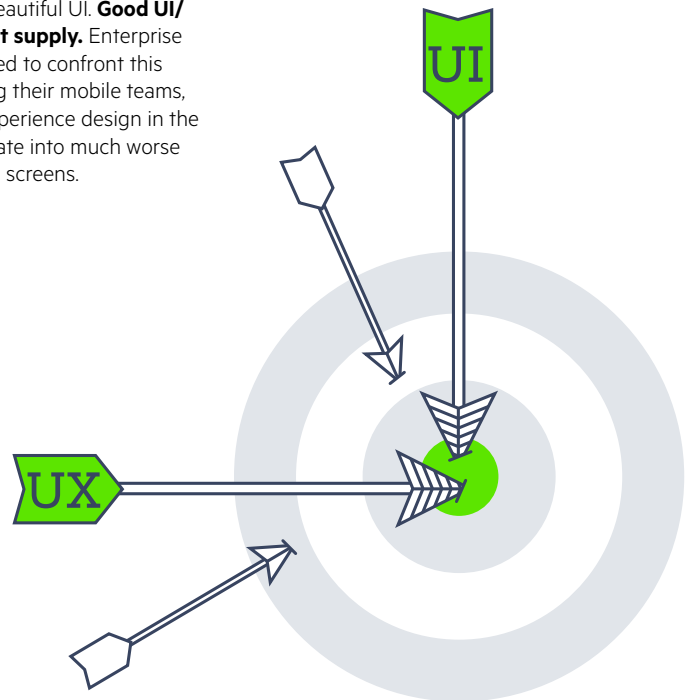
What's the Role of Reusable/ Pre-Built Components, and How Should Developers Use Them?

Mobile development depends on the use of pre-built components and widgets. Developers will share UI and UX components differently from the way they would share an HTTP call to go get data. For each component, the development team has to ask itself, will this be shared across the 300 business apps we are going to build, or between the website and its complementary app?

The next question is, how do these components get deployed? A piece of code, like an app, may be updated frequently, so it has to be pushed in unison, in each update, to all the apps that incorporate them.

JQuery is easy to reuse, because it is a definitive library of actions and UI controls that the developer uses at will, whereas calling an API that is dependent on a signature is something that could change with every release.

Enterprise teams typically have more people who are effective at writing business logic than those who know how to write a beautiful UI. **Good UI/UX developers are in short supply.** Enterprise development managers need to confront this issue early on when forming their mobile teams, because ineffective user experience design in the web world is likely to translate into much worse user experiences on mobile screens.





5.

ARTFUL TESTING WITH USER IN MIND



ARTFUL TESTING WITH USER IN MIND

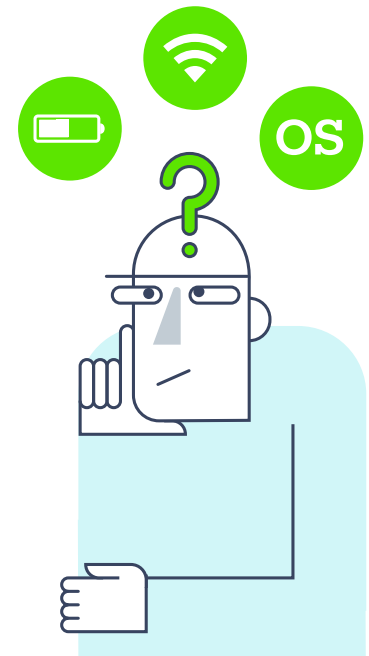
What is the most effective approach to mobile app testing? At this stage, approximately five years after the advent of mobile as a mainstream platform for applications, the jury is still out.

In web application development, the principal challenges have been browser related. Developers needed to be sure the application would run in Internet Explorer, Firefox and Chrome—all of the versions of those browsers in use—that they would handle AJAX effectively, and a host of other common browser issues. Mobile presents us with a new set of concerns, including:

- Battery life
- Signal: is the user connected to WIFI or not, and what is the signal strength?
- The hundreds or thousands of devices accessing the app

Although there are only three major operating systems for mobile—iOS, Android and Windows—there is nonetheless huge diversity in mobile environments. For years, iOS was quite uniform, but now it is becoming increasingly fragmented, as Apple rolls out more devices. The Android user base, with multiple vendors, models and configurations, is enormously fragmented.

The challenge is principally this: **how do you test against all those devices?** With organizations adopting BYOD, the development team cannot avoid the need to provide functionality on literally thousands of different screens.





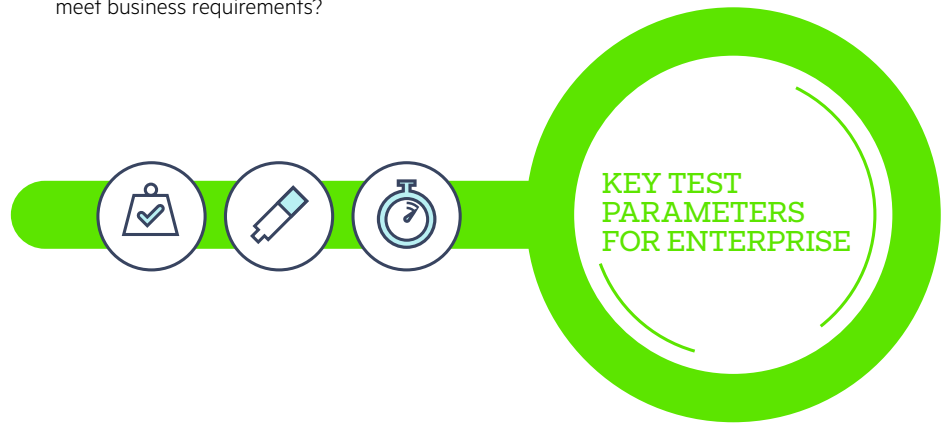
Continuous Integration (CI)—the accelerated rollout of platforms and applications in an enterprise—is yet another curveball. It was fairly straightforward to deploy and implement CI in the web world, but in mobile, developers need to build simultaneously in the Microsoft and Mac environments; your shop may be building Android apps in Visual Studio and Eclipse on PC, while building iOS apps on the Mac, meanwhile racing to keep the functionality in sync.

What Are the Key Test Parameters for the Enterprise?

The three principal areas of concern for mobile apps are:

- **Functional:** Does the app access the data, execute the functions and provide the User Experience required by the business? You want to eliminate functional defects on all platforms.
- **Performance:** Is the app fast and responsive, and does it complete every transaction, on all devices?
- **Load:** Can the app handle enough traffic to meet business requirements?

The rationale for functional testing is straightforward. What has changed in the transition from web to mobile is chiefly the diversity of devices and form factors on which the functionality must be assessed [see Chapter 3](#)





Power

There are additional native attributes of mobile devices that affect app functionality. A good example is power; mobile devices almost invariably run on batteries. Developers build in features to control the way the application behaves as the battery runs down. This is not simply a question of performance. Typically, the processor onboard the device will scale down to conserve battery as it approaches a critical point, and that can affect app behavior as onboard services become unavailable to it.

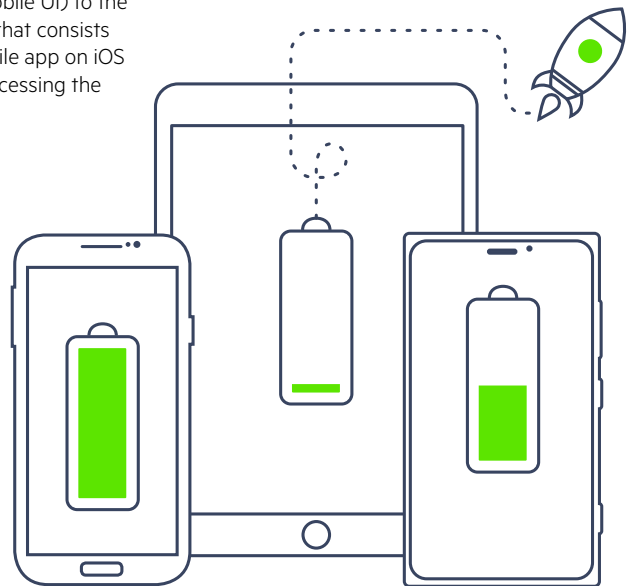
Battery life is handled by the OS, but there may be OEM customizations on certain devices; QA cannot assume all Android devices will handle battery degradation in the same way.

Performance and “Load”

Performance testing is critical, as performance lags can get in the way of adoption. Performance also can be affected by conflicts with other applications that are running concurrently (especially on Android devices).

Load testing involves creating scenarios driving usually “headless” traffic (traffic generated outside the web browser or the mobile UI) to the app. You may have an application that consists of a mobile app on Android, a mobile app on iOS and a web-based application all accessing the same enterprise data.

Analytics may tell you that 50 percent of the traffic is coming through on the web and 25 percent on each of the mobile OS apps. People are browsing through products, logging into their accounts; perhaps 2 percent of the users are filling out an RMA form. Scenarios like these capture traffic patterns.





The testing protocol is to construct such scenarios and assign virtual users to them. In one scenario, perhaps 10 percent of the iOS users are browsing the product pages. So, the test will designate 10 percent of the 1,000 virtual users allocated to the test for that process.

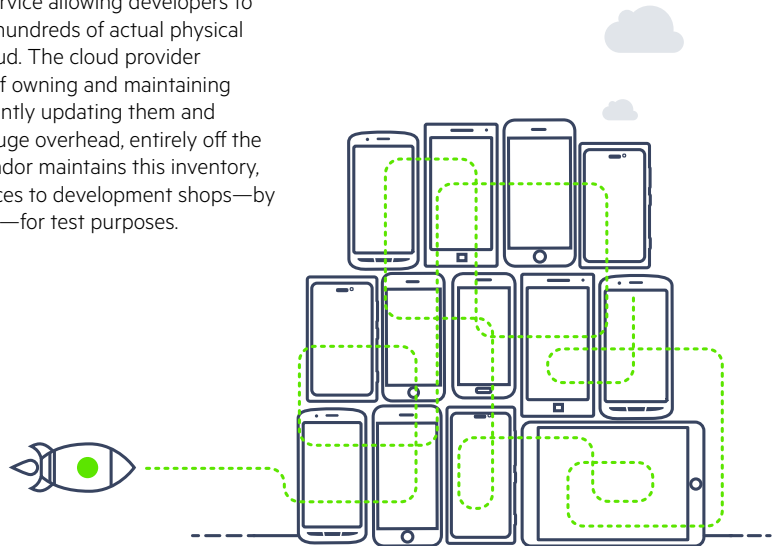
The test will do the same for the different scenarios and different platforms. You're testing how the app behaves under different loads, and whether there is a breaking point.

There are multiple reasons for this kind of testing. One application owner may say, "We're running a Super Bowl ad and we need to know whether we can handle a million users on the site." Or the developer may be creating an application and needs to know whether it will scale to some specific volume of users.

What Can Enterprises Do to Simplify Cross-Platform Testing?

It is impractical for an enterprise QA team to maintain a closet full of mobile devices for testing, given the diversity of devices and the rapid pace of technical evolution. The business case for outsourcing mobile testing is very compelling.

A powerful, recently developed option is the device cloud—a service allowing developers to test their apps on hundreds of actual physical devices via the cloud. The cloud provider takes the burden of owning and maintaining the devices, constantly updating them and maintaining that huge overhead, entirely off the developer. The vendor maintains this inventory, and rents the devices to development shops—by the minute or hour—for test purposes.





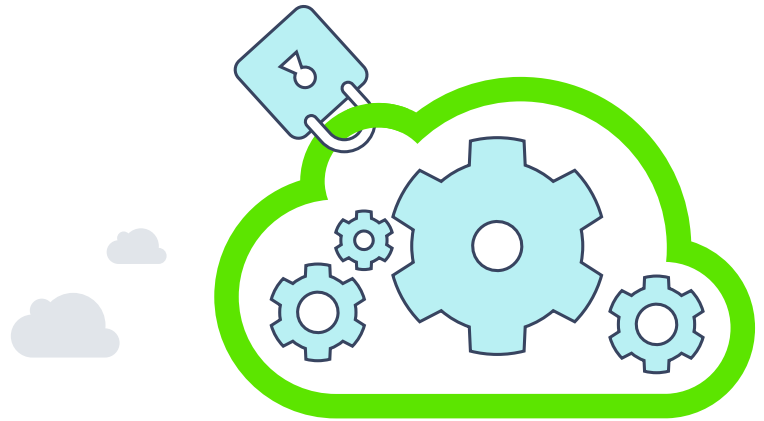
Maintaining a device cloud requires a complex infrastructure. There are security issues to address, not all of them related to data transmission. For example, is the test data from one user completely wiped from the device after the test is completed, before another user rents that device?

For organizations with very sensitive data or requirements for highly specialized devices, it could be worth the overhead expense and effort to set up an on-premises device cloud inside the firewall and run by the enterprise IT department. However, such an infrastructure is difficult and expensive to maintain for an organization committed to BYOD.

It may be feasible to **focus on a core set of devices**. Analytics can provide insight as to what devices users are bringing to work. The core set may be 30 different iOS and 20 Android devices. Devices beyond that set—the outliers—may be used in the enterprise rarely or sporadically, and it may make sense to outsource testing for just those devices to a device cloud maintained outside the firewall.

Test Scripting and Automation

In an ideal world, testing is broken down into logical components, much like the code itself. You are not conducting a single test across multiple platforms. The test script describes a series of functions—login to the application and execute a series of actions—without specifying the OS or the device. Then OS-specific details are added to the test script, but the scripting separates the find logic from the actual test logic, for easier maintenance.





Test automation is becoming increasingly popular but it is still in its infancy. There is no established leader in QA automation for mobile. It is difficult to say how much mobile app testing is automated but it's significantly lower than web application testing.

What Skills Do Enterprise Mobile App Testers Need?

Mobile app development teams tend to be smaller than web development teams, and frequently the developers themselves are responsible for testing. Many enterprises are just beginning to move some of their QA resources over from web development to mobile. They're just discovering what pieces of the apps can be tested through test automation, and what pieces still require manual testing.

First and foremost, a tester should have a solid understanding of the application. That has a lot to do with the tooling that is being used. If the app is heavily dependent on JavaScript, then JavaScript knowledge is a requirement for the tester. The other obvious requirement is a background in testing.

Testers often see QA as a path to a career in development. But this is not realistic for everyone, and there is an alternative path to career advancement within the QA discipline. For many people, a realistic goal is not to transition into coding but to mature into a first-rate QA engineer.

Collaboration Between QA And Development Is Critical

The team may be made up entirely of developers, in which case one or two of the developers are dedicated to testing, or at least to writing the automated test scripts. A lot of the automated test solutions are code-centric—they actually require some coding.





Alternatively, you may have teams that include non-technical testers—typically individuals who came up through the support function in the enterprise and whose career path has taken them through QA. Some of them are quite proficient with test automation tools, but there are always edge-case scenarios in which you have to get a developer involved to write some test automation code. At this point, however, the involvement of non-developer QA people is still more common in web application development than in mobile.

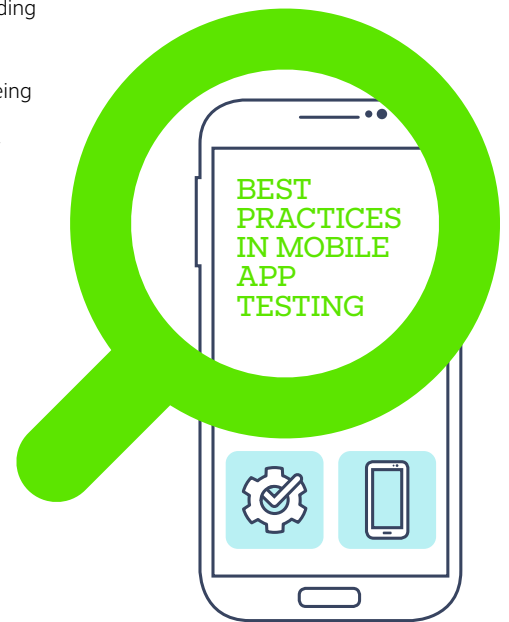
Collaboration is essential—once the automated test is created, it is important that the non-technical tester be able to take that test and run with it, without continually burdening the developers to build new test scripts.

QA has an essential role in Agile and Lean development organizations. It is important that testers be embraced as part of the team for each project, and have roles in each sprint. There should be user stories for the test cases, and for the purpose of Scrum, this should be treated like any piece of development.

Best Practices in Mobile App Testing

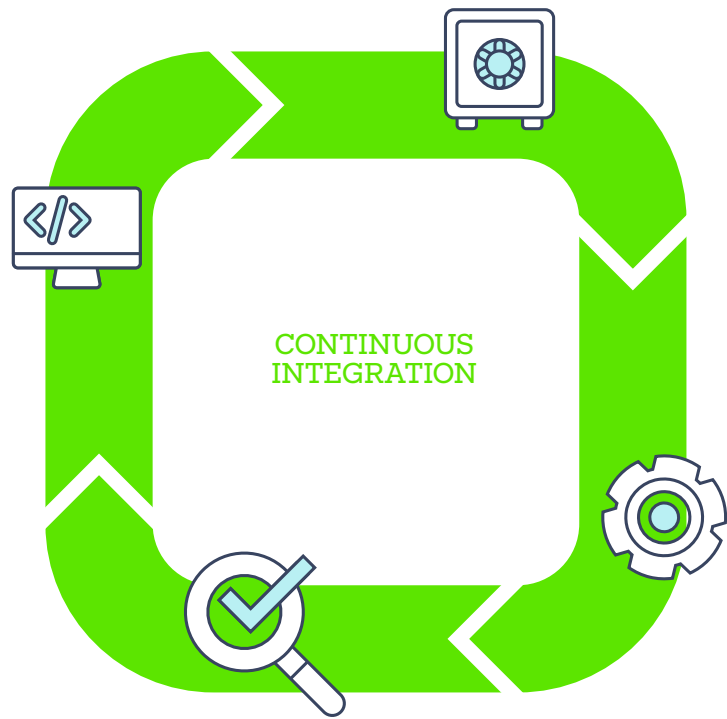
General principles emerging in the mobile QA discipline include the following:

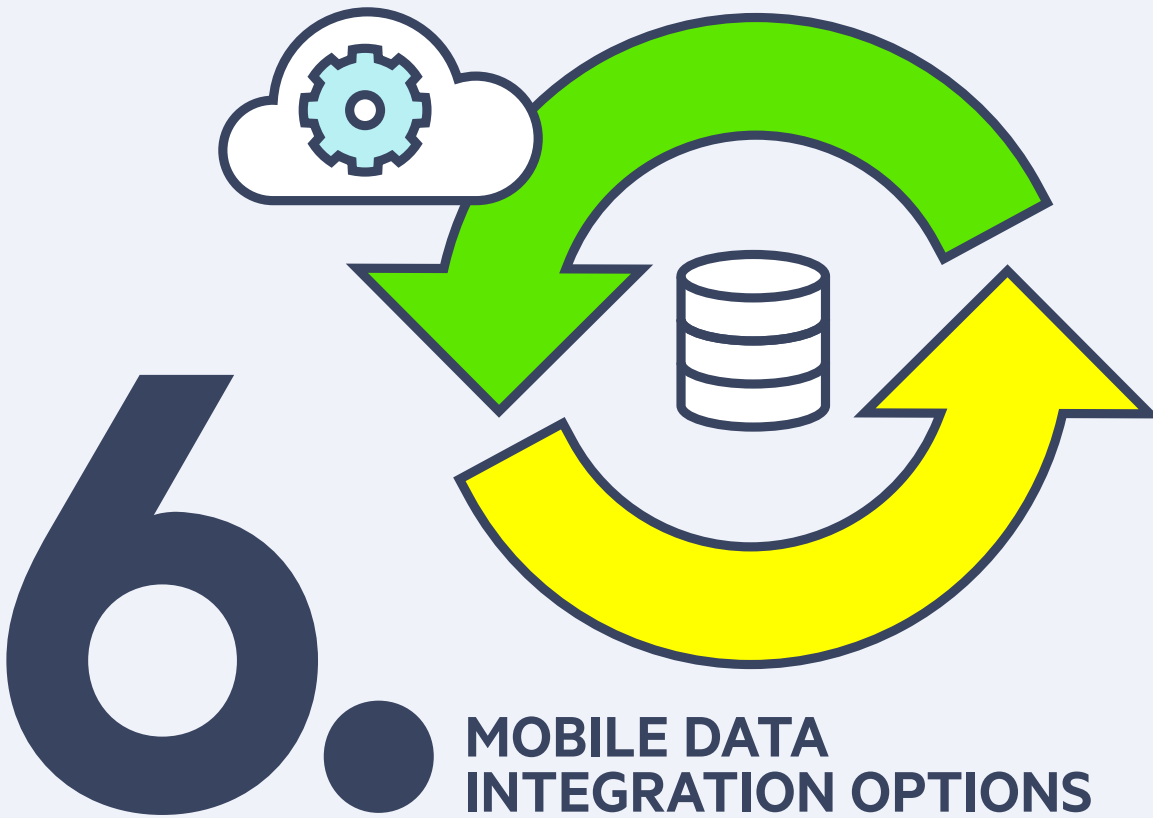
1. Know when to automate and when not to:
In edge-case scenarios that are difficult to script, manual testing often is more efficient than investing a lot of time in coding automated test solutions
2. Know the devices for which the app is being developed, and the analytics for those devices, as well as all the form factors for those devices





It is a challenge to implement **CI** in mobile. But more and more solutions are available to support this. As an additional best practice for CI shops, enterprises looking for a testing solution should be certain that it can be incorporated with their build server (Jenkins, TFS, Bamboo), so that they can build, deploy and automate test cases. CI challenges are not unique to mobile, but the problem of building and provisioning to 300 devices on two or three operating systems is unprecedented.





MOBILE DATA INTEGRATION OPTIONS



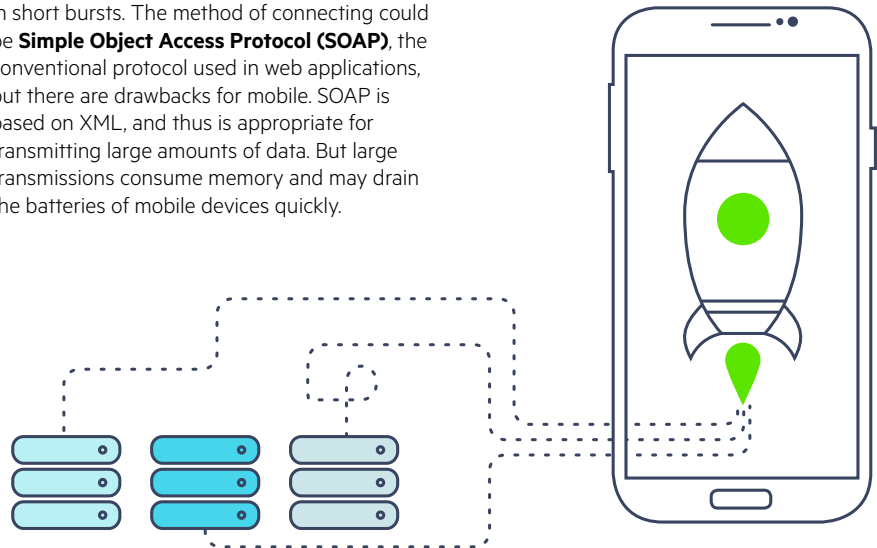
MOBILE DATA INTEGRATION OPTIONS

A mobile app does not exist independently of the backend data. There is a local UI, and it conventionally connects with the backend data repository through a middle layer. An enterprise is committed to a set of existing backend systems—Oracle, SQL Server and the like. The challenge is to construct the middle layer so that it consumes the backend data and exposes it through an API, enabling it to be used by the mobile app.

Enterprises typically have limited resources for middle-layer development. Options such as Backend as a Service (BaaS) enable the enterprise to identify the backend data and the API to be exposed (or services that can be exposed) to the mobile application, without writing code. In other words, the enterprise team can move from implementing and supporting the data connection to simply configuring it.

Data Transmission Approaches

A challenge in mobile connectivity is that backend systems must handle requests from thousands (sometimes millions) of mobile users, all connecting to the data, not continuously but in short bursts. The method of connecting could be **Simple Object Access Protocol (SOAP)**, the conventional protocol used in web applications, but there are drawbacks for mobile. SOAP is based on XML, and thus is appropriate for transmitting large amounts of data. But large transmissions consume memory and may drain the batteries of mobile devices quickly.





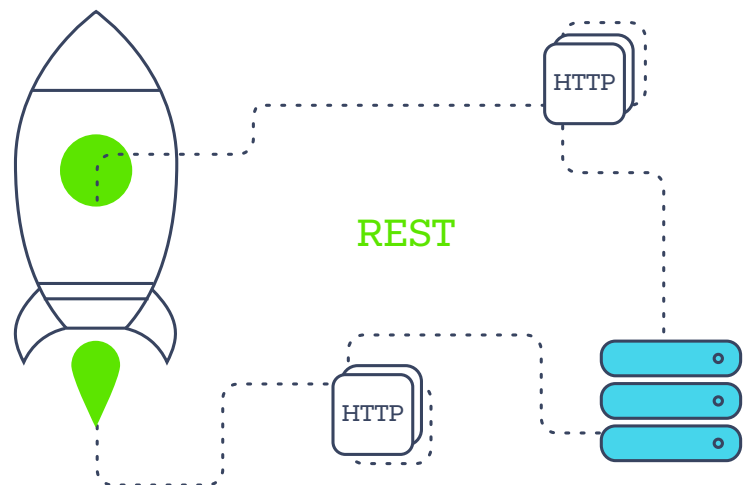
Conveying data in smaller increments makes more sense for many mobile scenarios, and the alternative protocol, **JavaScript Object Notation (JSON)**, is likely to be more efficient. Some apps are designed to use both, converting existing data interchange to JSON on the fly.

In recent years, web development teams in pursuit of simplicity and faster data transmission have adopted RESTful approaches—connectivity that takes advantage of **Representational State Transfer (REST)**, a lightweight architecture in which all data posting and editing operations are handled via HTTP calls. Mobile development objectives are similar—rapid, simple, reliable data interchange—so it will come as no surprise that REST is standard in mobile development.

Discontinuous Connection

Connectivity between a mobile device and the Internet is not constant. Consequently, any mobile app developer needs to consider the options for storing data, at least temporarily, on the device. The user wants to continue using the app for some operations, regardless of whether he is connected to the Internet. At a minimum, the app needs to present the user with an interface indicating that some operations are not available until he is reconnected to the network, while still providing basic functionality.

This requires offline storage on the device, and automatic syncing of the information between the device and the backend data repository in the cloud.





Among vendors of mobile application development solutions, data synchronization is one of the biggest aspects of the “story,” especially for enterprise customers. Even for a consumer-oriented app, the loss of functionality when offline is a serious concern if it degrades the experience for the user, because it can reduce the user’s satisfaction with the app and reflect badly on the brand sponsoring it.

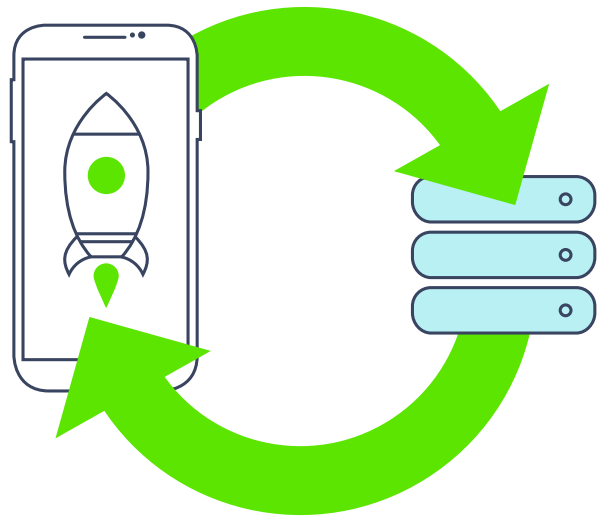
But for enterprise app development, effective synchronization is even more critical, as users will be depending on the app to provide access to enterprise data for vital business processes.

Developers are only beginning to see maturing approaches to data synchronization—there is no easy way to do it today. Loss of network connectivity is only one of the issues; synchronization can be affected by conflicts between multiple users and multiple apps accessing the same data store.

Deep expertise and effective tools for synchronization will be one of the most differentiating attributes of competing vendors of mobile app development platforms and services for years to come. And for enterprise development teams considering the in-house coding of their own connectivity solutions, this will be one of the most daunting challenges.

Flexibility in Synchronization

Whether you are connecting the app to a SQL server, an Oracle server or to Salesforce, it doesn’t really matter. The important architectural principle is to design your synchronization functions at a level of abstraction so that they treat all of those backend data stores the same way.

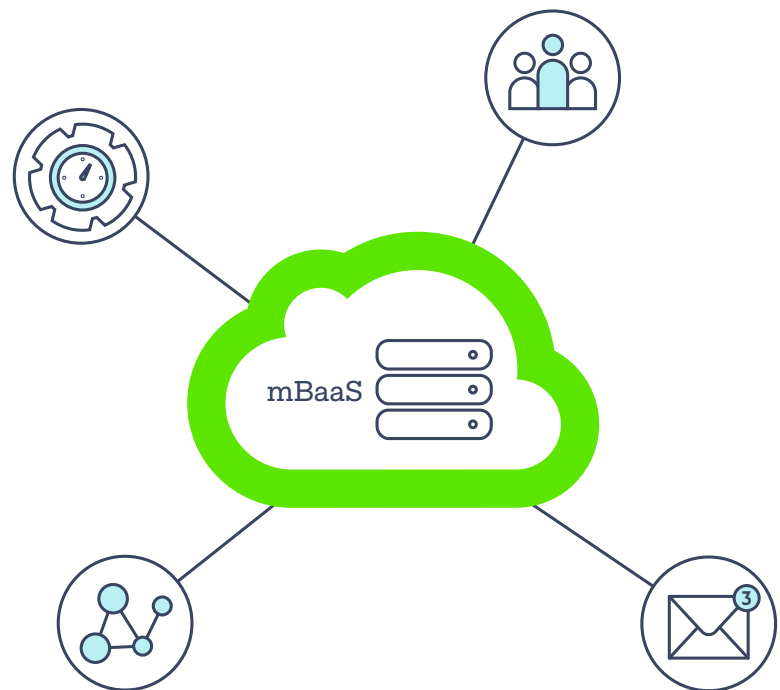




This flexibility is critical; if the development team designed its data connection for a unique, specific data source, it would have an entirely new problem when presented with the need to connect a new app to an entirely different data source—a very common enterprise scenario.

It's still conventional to ask enterprise developers to implement their own services on top of backend systems, exposing the data to the specific mobile devices that need to consume it. A benefit of this approach is developers get full control of those services; the downside is that the approach offers limited scalability within the enterprise—it's a model for enterprise development that is only sustainable with a relatively large and productive development staff.

Captive development might make sense for certain small- to medium-sized businesses with limited numbers of applications to build and maintain. For many enterprises—perhaps most—it makes more sense to outsource the connectivity solution—essentially renting the service from a **Mobile Backend as a Service (mBaaS)** vendor, or licensing middleware from an existing vendor's library of tools. Either scenario reduces the in-house engineers' role to configuration, as opposed to original development.





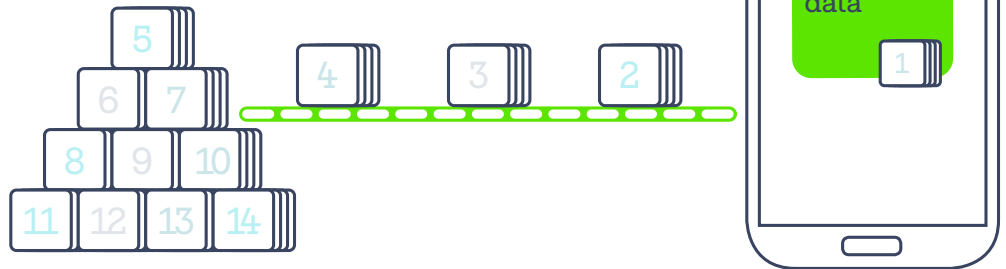
Key Considerations In Connecting to Enterprise Data

Adopting a vendor's mobile middleware has the advantage of enabling the engineers to configure access to multiple enterprise data sources simultaneously. The commercial middleware will be optimized for mobile consumption, incorporating data compression and other services designed specifically for mobile.

Generally, it is not practical to reuse services designed for the rest of the enterprise architecture when moving to mobile. The payload those legacy services deliver to the end user is too big for phones or tablets. To reconfigure those services to deliver data in smaller increments and adapt to frequent interruptions in connectivity would not be worth the effort and expense.

A common technique is **paging**—chunking of transmitted data into small increments (pages). The design is such that, while the repository may contain a million records, the retrieval protocol is simply to get the first 50, then the next 50 and so on. Paging is not new or unique to mobile, but it is highly practical for the purpose.

An alternative is to store large volumes of data on the mobile client, allowing the user to work offline, but this is not practical on all devices, some of which will be hampered by limited onboard storage. This approach cannot be implemented in HTML, because HTML imposes a limit of 10 Mbytes of local storage on the device.





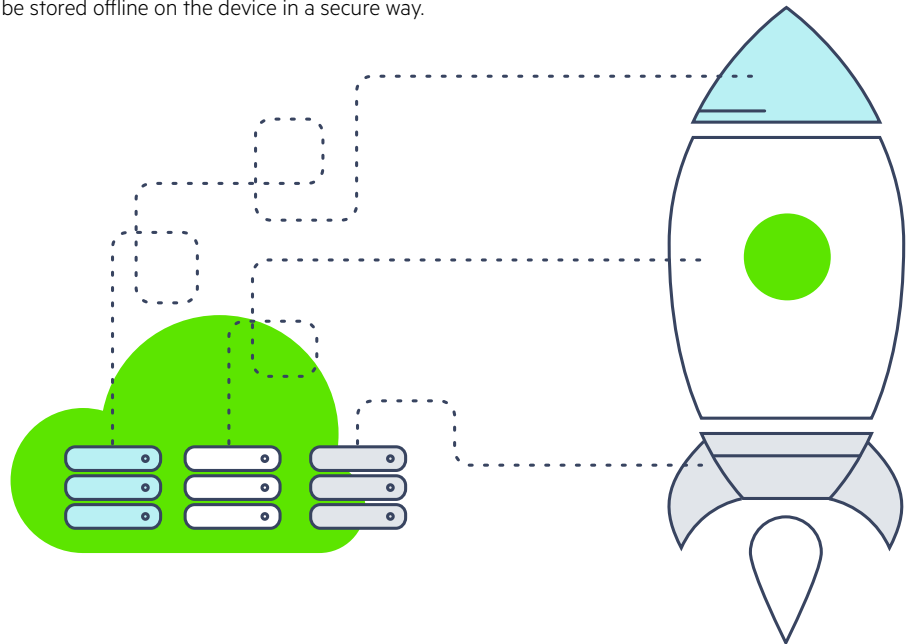
The vendors have standardized ways to access data from the popular data stores, and generally the enterprise development team can feel confident that a commercial connectivity solution is efficient and secure. What is not standard, however, is the approach to drawing data from multiple data stores, combining these data streams for a single application consuming the information. A typical way of accomplishing this is via **data virtualization** or **data federation**, in which data from multiple sources is combined on the fly into a single entity that makes sense to consume using the app.

For example, an enterprise client of Telerik recently had a list of employees stored in SQL server, and wanted to be able to determine in a query who the supervisor was for any individual—the representation of the company hierarchy was stored in Active Directory. The goal was to combine the data from the two sources into a single entity to be consumed by the device.

This application of data virtualization currently is a read-only approach. Writing back to the two data sources is much more challenging, because the developer has to understand the logic by which the two sources came together. But read-only is sufficient for a high percentage of use cases.

Encrypting Data In Transit

If the data is very sensitive and requires a certain level of encryption, the enterprise might have an incentive to develop its own connectivity solution. Some of the vendors offer encryption on the device, but encryption is not subject to a mature standard, and some enterprises may have security policies for which off-the-shelf tools fall short—particularly if the requirement is that data be stored offline on the device in a secure way.





Vendors such as Telerik provide data synchronization components that encrypt data by default. Third-party encryption tools, including open-source tools like Zetetic LLC's SQLCipher, also are available to provide encryption as an add-on service.

Off-the-Shelf Connectors—Pros and Cons

Off-the-shelf development tools can provide connectivity to the most broadly adopted data stores, using middleware optimized for mobile. This includes not only connectivity for completed apps, but some vendors can provide connectivity at the widget level—the component, designed to manage a very specific service within multiple apps, comes already set up to talk to multiple popular data stores, with state-of-the-art synchronization for mobile.

This includes components that have nothing to do with the UI—building blocks common to multiple apps that handle deeply embedded functions. An example is **Responsive Image Services**, in Telerik Platform. This function takes note of the app user's device in serving up images, so that the data store will automatically resize an image in the server; it won't send a 10MB image to a small phone that can only practically display a much smaller image, for example.

Like any approach in development, the purchase of off-the-shelf components involves a tradeoff. This decision will lock the enterprise into a relationship with a vendor—one that should not be entered into casually.

If the vendor frequently changes course or discontinues support for components on which the enterprise depends, there is risk. Given the current consolidation trend in the market for mobile development tools, such risk must be considered.





It makes sense to choose a vendor with a track record of strong execution and a comprehensive strategic vision that extends two to five years into the future. The economics of such a relationship are compelling, but the partner will be contributing important components to your mobile strategy—it's a high-stakes decision.

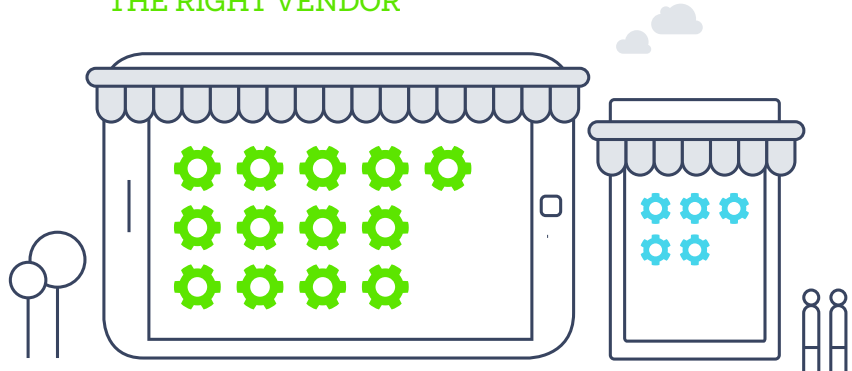
The optimal approach may be to partner with more than one vendor, choosing partners with complementary strengths. Both Azure and Amazon are likely to survive in the market for years to come, and could be effective partners for a vendor with a better story in offline synchronization, as an example.

Organizations adopting mobile have had to accept that data will persist on those devices. Their approach has been to apply corporate policies at the device level—always insisting on a secure passcode for the device. There are policies regarding the strength of the passcode. And the policy may be that if the user fails three times to enter the correct passcode, the system can wipe the data from the entire device. Such policies can be administered remotely. If the company has former employees who've used corporate data, the IT department can wipe specific apps from the devices of former employees.

Service Levels

Connectivity solutions come in ranges of price and functionality. At one end is a free option, available without a license fee, that does not include the security features—there is no support for the secure service layer to access data over an HTTPS connection, but just a straight HTTP-to-HTTP connection. To use it, you need to pay for the backend services, where the security will be in place.

SELECTING THE RIGHT VENDOR



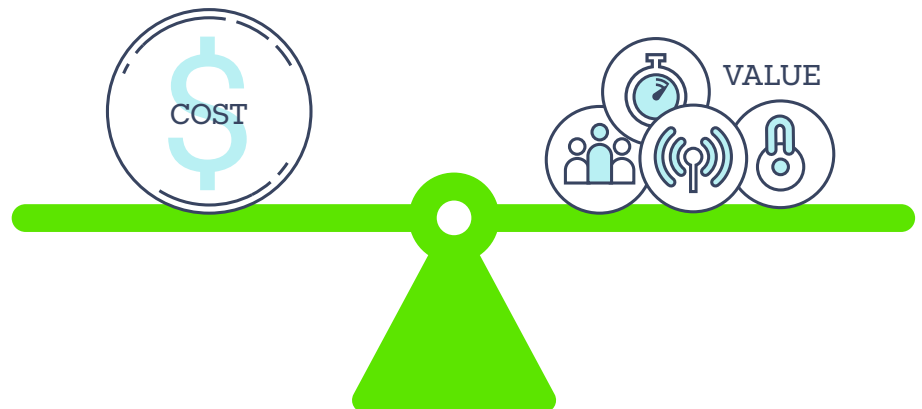


If the app requires connection to Active Directory—if you want your users to log in with LDAP—you'll likely need a high-end service package. Otherwise, the app will use built-in security, and users will not be able to log in with authentication from Active Directory. Also, most solutions come with the service of cloud data storage. You may only be able to store data in a proprietary data store inside your firewall with the high-end versions of the service.

The Value Equation

The importance of an app to the enterprise may not be in proportion to the number of people using it. A consumer app that is used by millions may have a lower value to the organization than a business app used by 10 employees. So from a value perspective, it does not make sense to price services on the basis of scale, and vendors generally will provide the same commitment to performance, regardless of scale.

Teleryk requires that if a client's app receives more than half a million requests per month, the data is moved to its own location on Teleryk servers, where the company can assure optimal performance, not just of the client's app but of its neighbors within the infrastructure.



7



MANAGEMENT AND APPLICATION SECURITY



MANAGEMENT AND APPLICATION SECURITY

Application Management and Device Management are concepts that can be referred to collectively as **Enterprise Mobility Management**. This term covers all of the important aspects of mobile security—Mobile Device Management (MDM), Mobile Application Management (MAM) and Mobile Content Management (MCM). The major solution providers have broadened their offerings to cover three aspects.

Mobile Device Management

MDM enables the enterprise to maintain an inventory of hardware devices. MDM differentiates devices by the OS and its capabilities. The leading vendors maintain very close relationships with the OS originators, and typically ship the updated version of MDM software the same day a new release of the OS is shipped. It is typical for enterprises to upgrade to the latest version of iOS in less than two weeks, in part because they can get support from the MDM vendor the same day as the OS release.

An important driver is the BYOD trend—which creates a critical difference between mobile and desktop support. You don't have a choice as an enterprise whether to adopt the latest version of the OS when it comes out. The users are making that decision for you. In the desktop world, by contrast, an employee would rely on a corporate-owned computer controlled by the policies of the company, and enterprises might resist OS or browser upgrades for months after release, or skip releases entirely.

All MDM platforms are not quite the same. The standards are not yet clear. But this is largely irrelevant to the software developer, since MDM only concerns devices. For MAM, there may be different approaches to software containerization. If the software is containerized, the software developer should have no real concerns. They are replacing unsecured code with a wrapper of secure code. If your application code is unsecured for some reason, the container wraps it in code that is optimized and secure for mobile.



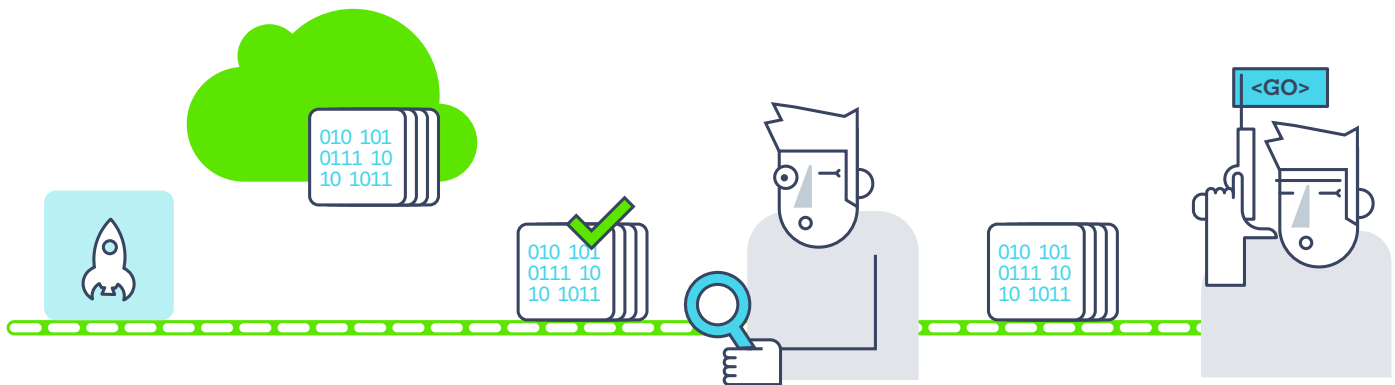


Mobile Application Management

On the other hand, MAM providers offer specific **Software Development Kits (SDKs)**—if the development team chooses to work with the SDK for a given project, there will be no container. What the developer does have is much more granular control. Using the SDK, he has secure access to the behind-the-firewall services of the company, and the app will be designed to store some data on the device. Using an SDK results in an application that uses the data and services within the security policy of the company.

The choices are not mutually exclusive. You can have both containerization and SDKs. But you can make the choice depending on the engineering resources you have in the enterprise. An enterprise with development depth, creating applications for its own purposes, will likely prefer the SDK approach; if you have third parties developing applications for you, containerization probably may make more sense.

Airwatch and MobileIron, two leading MAM vendors, work similarly: when the developer has completed the building and QA for an app, he uploads the binary file of the application to a web portal, and the vendor wraps it. The process is simple; any administrator can do it.



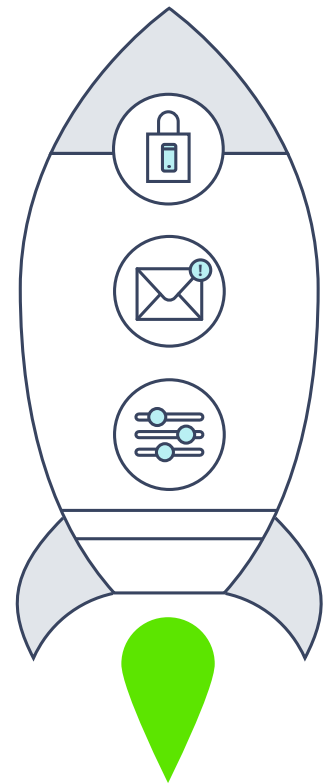


One of the MAM system's capabilities is that of serving as an **application store**. MAM might have a range of other management functions and enforce various policies, such as specifying that a specific set of applications have specific properties (for example, they always run under a particular security protocol, such as DTM). MAM systems can alert users to the existence of a new version of an app. Some individual on the development team should have the permissions to compile the application and store it to the repository, and be designated as the one responsible for doing so.

A typical enterprise can have its own app store. It provides very good control of distribution to the end users, so that the application can never leave the security borders of the company. The Apple or Google Play stores are secure, but providing an enterprise store enables the company to specify its own security policies, and the development team can easily define the roles within the business unit and map those to roles defined by app store. For example, the HR team might only see the HR applications, or the travel team might only see the travel expense application. This is very similar to software distribution systems that have been used for many years for desktop applications.

Managing Software Components

Generally, MAM is used for managing complete apps. As we have seen in earlier chapters, mobile apps generally are made up of smaller components of code that handle specific actions and services, either developed in-house for reuse in multiple projects, procured from the Open Source community or licensed from a mobile development tool vendor. It's possible to use the MAM system to manage smaller widgets, as well, but this is atypical. The sharing of small components and widgets among developers usually is much less formal, through some kind of source control system (such as Visual Studio or Eclipse).

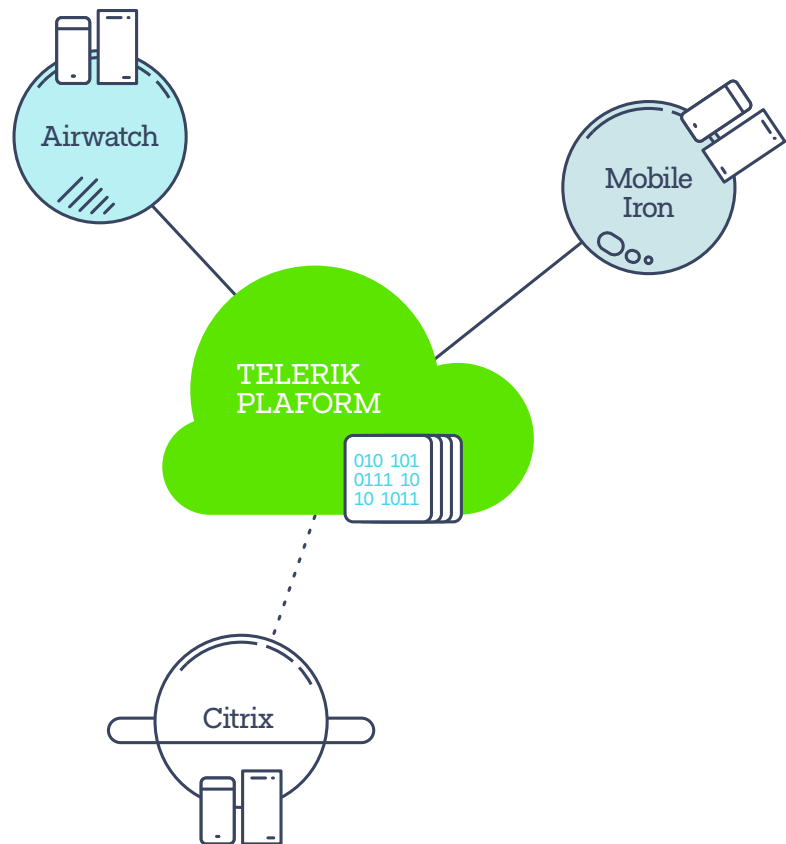




This code repository should be integrated with the MAM system. When you're building your application, you've completed an Minimum Viable Product, tested it and should be able to publish the binary files directly to Airwatch. Airwatch will then wrap the application for distribution to end users.

Currently, Telerik is integrating the SDKs of various providers into its platform, so that the developer can promote the finished application to the MAM system and then decide whether to containerize it. We are working with Airwatch (VMWare) and MobileIron. We will ultimately integrate with Citrix as well. Airwatch has the necessary APIs for integration, but MobileIron does not.

A best practice is to **constantly run integration tests** on anything the team has integrated into its mobile platform through APIs. If the enterprise has licensed a tool that provides an API, it should anticipate that the vendor will conduct similar tests, as well, but should test these connections continuously. APIs evolve rapidly, especially in the mobile development world, and they represent a potentially strategic dependency.





MAKING MOBILE ANALYTICS WORK



MAKING MOBILE ANALYTICS WORK

From the moment the development team releases a new app, that code is in the wild—in the hands of users, and beyond the control of the developers. Fortunately, you don't have to simply wish it good luck and move on to the next project. The app takes with it the eyes and ears of its developers, in the form of analytics built into the development platform.

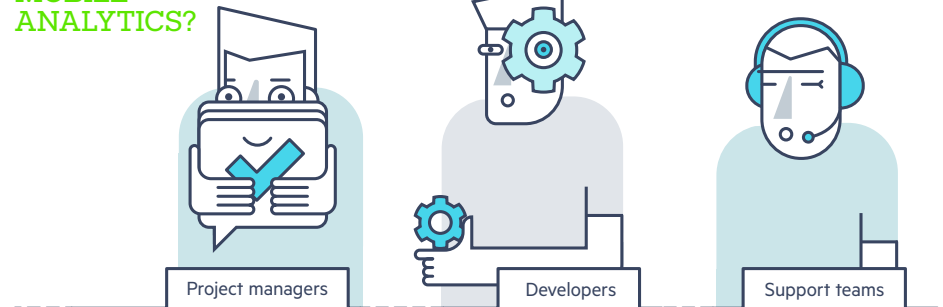
Who Needs Mobile Analytics Most in the Organization?

In the enterprise, the main consumers of analytics are the product managers and the development team. The developers are interested in the usage of the features, but also in seeing whether the application is performing well—exceptions and errors are of the greatest interest to development.

Many enterprise developers think about analytics too late in the development cycle, implementing analytics after the product has been released, and thereby losing insights in the hours after the initial deployment. It also is possible to start too early with analytics. It's best to implement them before field or beta testing. If you do a lot of refactoring, then you will be changing the metrics you collect inside the application.

Analytics of the sort that are of interest to developers rarely, if ever, are exposed to the end user. It is likely customers would not make much sense out of the data. However, some organizations will find it useful to Service Desk support people with access to certain analytics, to track performance of apps and mobile devices against service-level agreements.

WHO NEEDS MOBILE ANALYTICS?





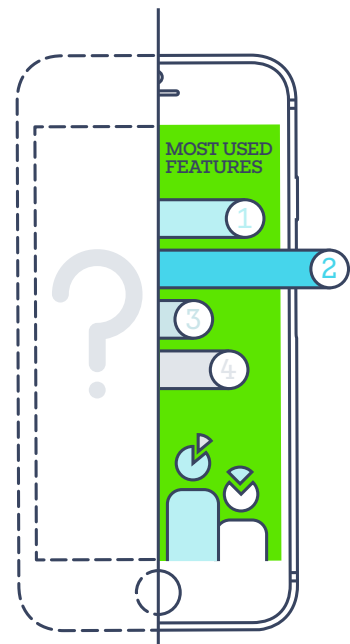
What Kind of Data Should Be Collected?

Analytics provide feedback on the way users access the app, and how often they use it. They can be quite granular in their content, advising the development team on which features are being used, how often they're being used and, conversely, which features are not used (so the team can better plan the next development cycle and make better roadmap decisions).

Analytics also can generate **insights into the users themselves**—the screen resolution of their devices, the operating system and other features of the environment. They can offer guidance on which features need to be enhanced, and which can safely be dropped in future releases. They can tip you off that it is safe to drop support for a little-used OS.

Other commonly used, high-value metrics generated by analytics include:

- **Feature-timing:** The system can track how long it took for the application to start up and show the initial screen, how long it took to process an image and so on. Analytics collect this data and send it to the backend servers, and the developers can see average times over a given interval on a dashboard designed for this purpose.
- **Screen resolution:** Developers know what screen size users have on their devices, how much RAM and other environment data. They know how often each type of device is used to access the application—whether it's an iOS, Android or Windows phone. This can be very helpful in allocating R&D resources to optimizing app functionality, UX and performance on these devices.





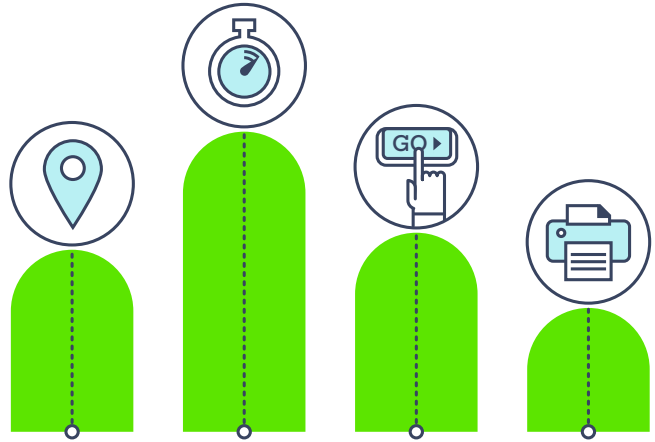
Analytics can be configured to track common features of apps across multiple business functions and technology platforms, as well as unique and esoteric apps for highly specialized devices. For example, in the Danish Post Office Department, package carriers use a mobile bar code scanner, and the application that runs it has analytics built-in.

This is an example of a very large enterprise with a very specific use case, and an application used only by one class of employees. They track how long it takes to scan, how was the mobile connection for that GPS location and so on. Package carriers don't give much direct feedback to the development team—nor, for that matter, indirect feedback unless something goes wrong with the scanner—so the analytics fill in the gaps.

What to Track...and What Not to Track

Analytics require configuration and return large volumes of data, so it is important to recognize that not all features, either of the app or the environment, are worth tracking. The team should decide beforehand which features are important to the business and prioritize tracking of those features. Generally, these are UI **features**—which buttons are clicked and which UI forms are shown to the user, how often a user printed something and so on. They are **business functions**—tangible elements of the User Experience, not individual widgets.

Analytics can be used to track application usage, not only by named users, but by anonymous users as well, by IP address, custom installation ID or an anonymous ID generated by the system. This can be accomplished privately, without collecting or sharing any individually identifiable user information.





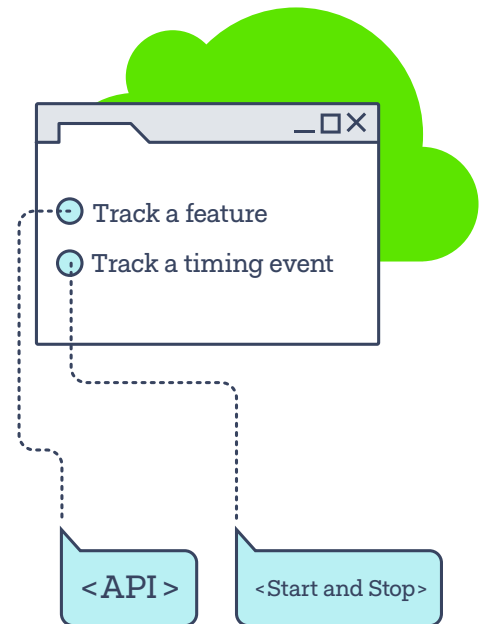
How Frequently Should Analytics Be Monitored?

Developers access analytics through browser-based portal, providing access to hosted backend data. The provider typically will enable users to subscribe to email alerts when certain exceptions or errors are encountered, and users can schedule regular reports at predetermined intervals.

Product Management might only look at the data once a month to see how new features are being adopted and how they're performing, to make decisions on future UI changes to improve performance or drive better adoption rates. Determining which features not to take into a future release of a product—especially moving from one platform to another—is one of the most effective ways to save money on development.

Users typically get a **monthly subscription for analytics**, and then embed, through an API, a simple library into the app. There are specific libraries for many different platforms—not only mobile devices but also for the desktop (Windows and MacOS). The developer enters a key in the library when she initializes it, and the tool starts collecting data. For example, to begin tracking the usage of selected features of the app, the developer would use an API called **Track Feature**.

Choosing features to track, based on their importance to the business, is something the team will do manually, so the decisions should be made in advance. To track a feature, you call an API; if you want to track a timing event, you call Start and Stop, and the system will track that interval.





The subscription includes all of the available metrics. It's based on the number of unique users you have. Typically an enterprise with fewer than 1,000 unique users can use a free subscription, but larger enterprises will need to go for the Professional subscription. Users can access data read-only, or they can be given rights to make changes. The account owner can assign users by name.

How Can One Ensure Security of Mobile Analytics Data?

It takes time to get development teams to adopt analytics. Years ago, it wasn't typical for developers to version-control their code; now version control is a standard tool. Without analytics, you're coding blind. Things can work in the development servers, but you don't know how they will perform in the customers' devices until you release a version and see what happens. Analytics give you instant feedback about incidents and problems.

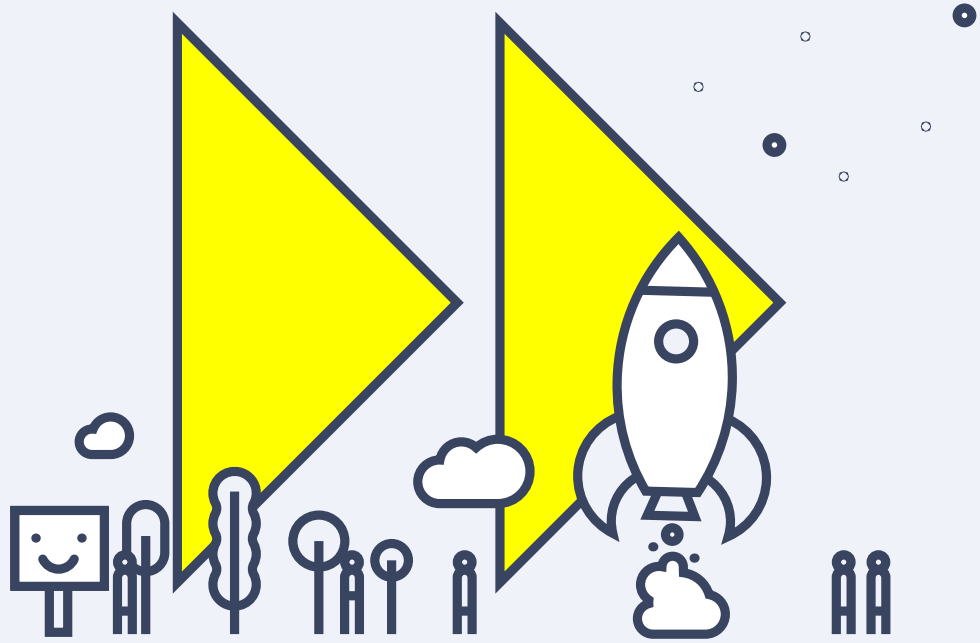
Commercial and Open Source Analytics Available

The issue is mainly to provide a backend service to manage the analytic data. At Telerik, we have multiple servers handling millions of messages per day; it is a major server installation and quite a complex solution.

Telerik has a dedicated server farm for analytics and stores the data there, but some customers prefer their own servers. An on-premises solution is an option that will be attractive to enterprises with apps accessing highly sensitive data. The tradeoff is the cost of maintaining the server infrastructure, including backup and restore, which really should be dedicated to analytics and not shared with other tenants. The cost of this is not trivial.



SUMMARY



SUMMARY



SUMMARY

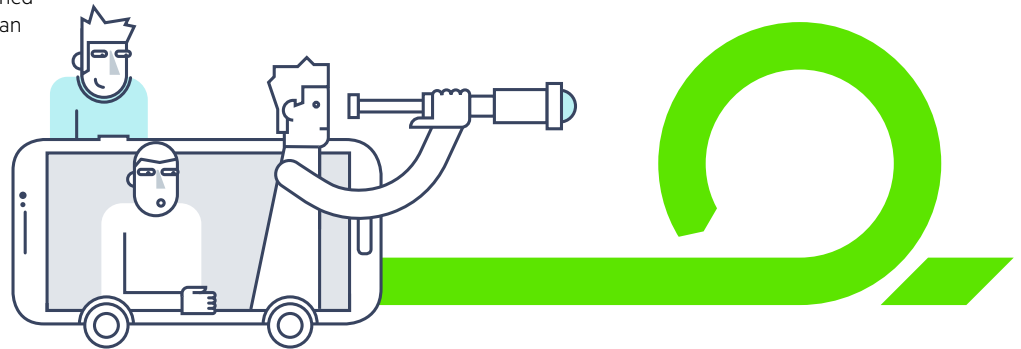
This eBook has sought to present a coherent picture of the challenges and opportunities ahead for an enterprise software engineering team that is beginning to develop the capacity for mobile app development.

If your development team has not already fully embraced **agile methodologies**, mobile will almost inevitably push you in that direction. The rate at which mobile technology is evolving, and the higher expectations about the quality of the user experience in mobile apps make agile methods essentially mandatory. “Agile,” of course, may mean strict adherence to a set of published principles, or selective adoption of agile or lean methods on a “use what works” basis.

A Degree of Centralization

You are likely to find that mobile development causes you to rethink the way you organize the engineering function in your enterprise. The emphasis on reuse of effective code and methods across multiple projects may make it attractive to centralize mobile development to a degree you have not done so in desktop or web development.

Mobile development also is very likely to lead you toward building from complete, mature code frameworks and off-the-shelf widgets, either open source or from vendors who specialize in such frameworks. Development teams already committed to service-oriented or software-defined architectures will have few issues with this approach.



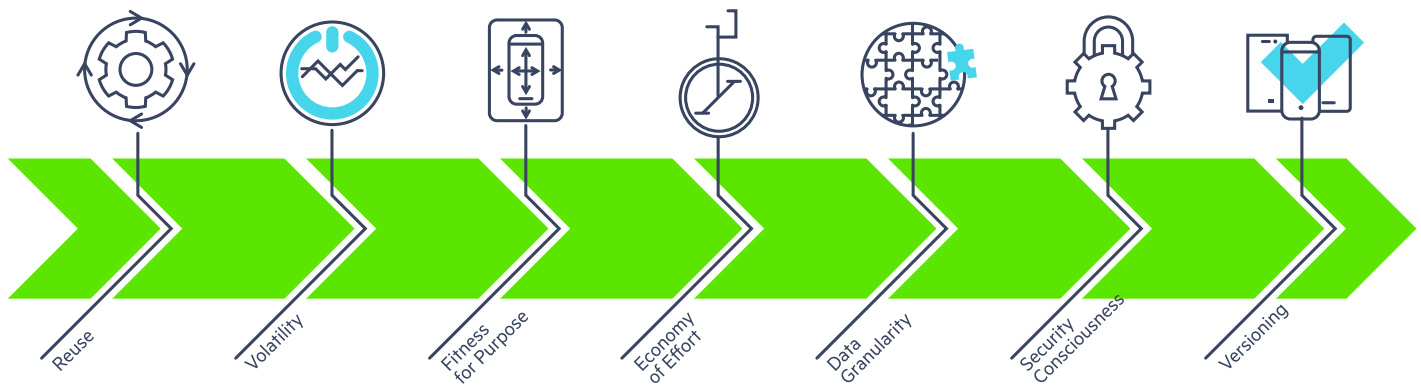


An important driver for centralized development and the use of standardize frameworks is the large number of devices and form factors for which a mobile app must be designed to work. Organizations embracing BYOD will deploy their apps to potentially hundreds of device types, with at least three important operating systems and thousands of possible configurations. While a certain amount of coding in native script may be necessary for an app to provide a satisfying user experience, building from a comprehensive framework will simplify the process of creating “responsive” apps delivering a high-value UX on multiple platforms.

Seven Key Principles

Once your mobility team and your tooling are in place, you will find that seven key principles will apply across most of your mobile projects:

1. **Code reuse**
2. The need to address the **volatility of network access** in your design
3. The importance of understanding the business problem to be solved by the app, to achieve **Fitness for Purpose**
4. **Economy of effort**, achieved through the choice of agile methodology, and the appropriate use of hybrid coding
5. Smaller data **granularity** along with the recognition that mobile apps access backend data in smaller increments than would be characteristic of a web application
6. Greater **security-consciousness** (because APIs in the mobile world are not hidden behind firewalls), including a default reliance on data encryption
7. **Versioning** and the recognition that at any time the app store serving up your enterprise apps is likely to contain multiple versions, all of which must be supported

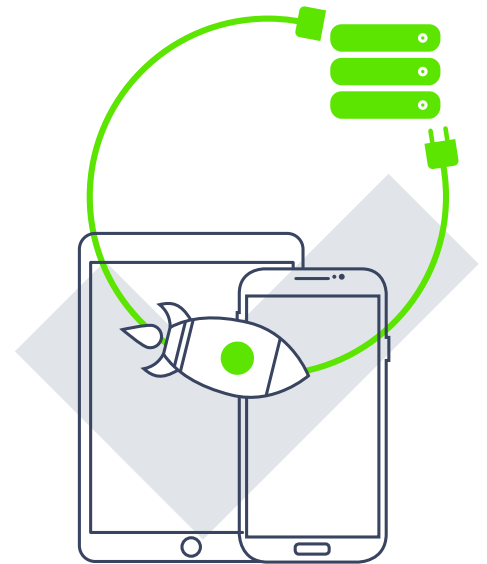


Mobile app testing will bring new challenges, as you must QA apps for a wide variety of devices and conditions—very large numbers of concurrent users, platform fragmentation, inconsistent network connections, power and so on. Mobile has led to the evolution of automated, cloud-based testing approaches that will be new to many enterprise development teams.

Backend Data Connections

Mobile may cause your engineers to rethink the way they connect apps to backend data. Conventional approaches like SOAP that assume a continuous network connection and downloading of data in large volumes may not work in a mobile context. Techniques like **JSON**, friendlier for situations where data will be accessed in smaller chunks, are preferred in mobile development, as will be the assumption that a certain amount of data will be stored on the device, at least temporarily, to enable the app to function offline.

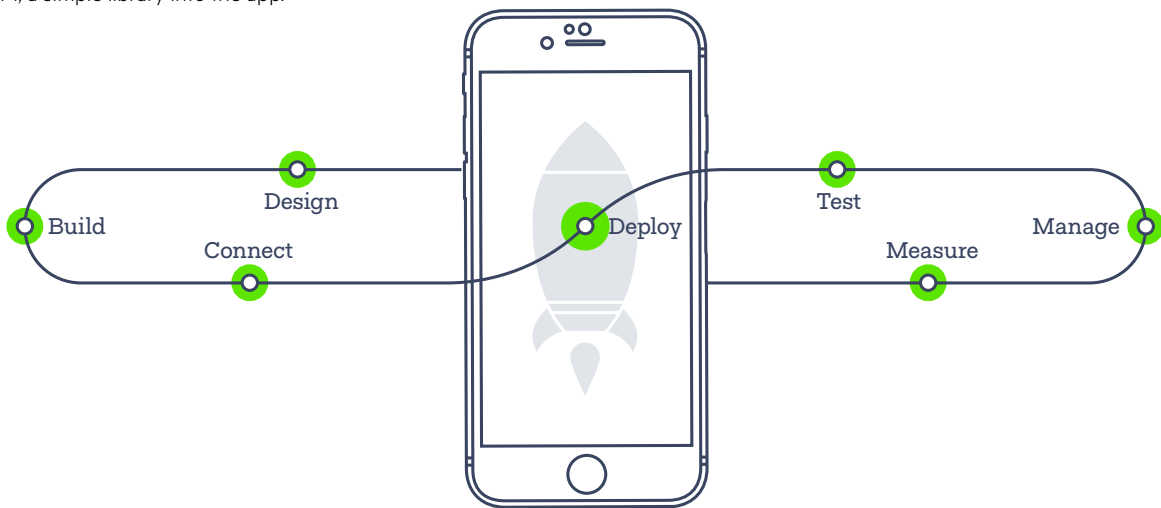
Application Management and Device Management are concepts can be referred to collectively as Enterprise Mobility Management. The major solution providers have broadened their offerings to cover these bases. **MDM** allows the organization to manage its hardware inventory; **MAM** is concerned with deployment of complete apps, and incorporates security features. A key concept is software **containerization**, in which functional code is “wrapped” in a code layer that is optimized and secure for mobile use.





Support for mobile apps is greatly enhanced by the incorporation of **analytics**, which feed back to the development team continuous streams of data about which features are being used, how often they are being used and, conversely, which features are not used. Analytics also generate insights into the screen resolution of users' devices, the OS and many other features of the environment. Users typically get a monthly subscription for analytics, and then embed, through an API, a simple library into the app.

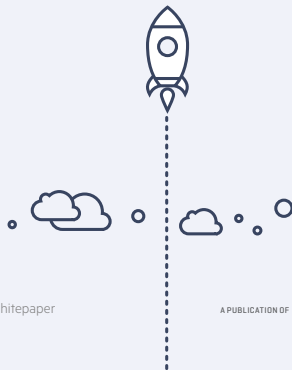
Clearly, the move to mobile development is an involved undertaking. We hope that you now have the beginnings of a roadmap for this crucial evolution.



LET TELERIK GUIDE YOU TOWARDS ENTERPRISE MOBILITY SUCCESS

Our completely integrated platform enables you to manage the entire application lifecycle from idea and development, to deployment, to measurement.

Try now



[Request a demo](#) to learn how Telerik can help with your mobility strategy.

Learn more about Telerik Platform and our suite of offerings. Visit us on:

