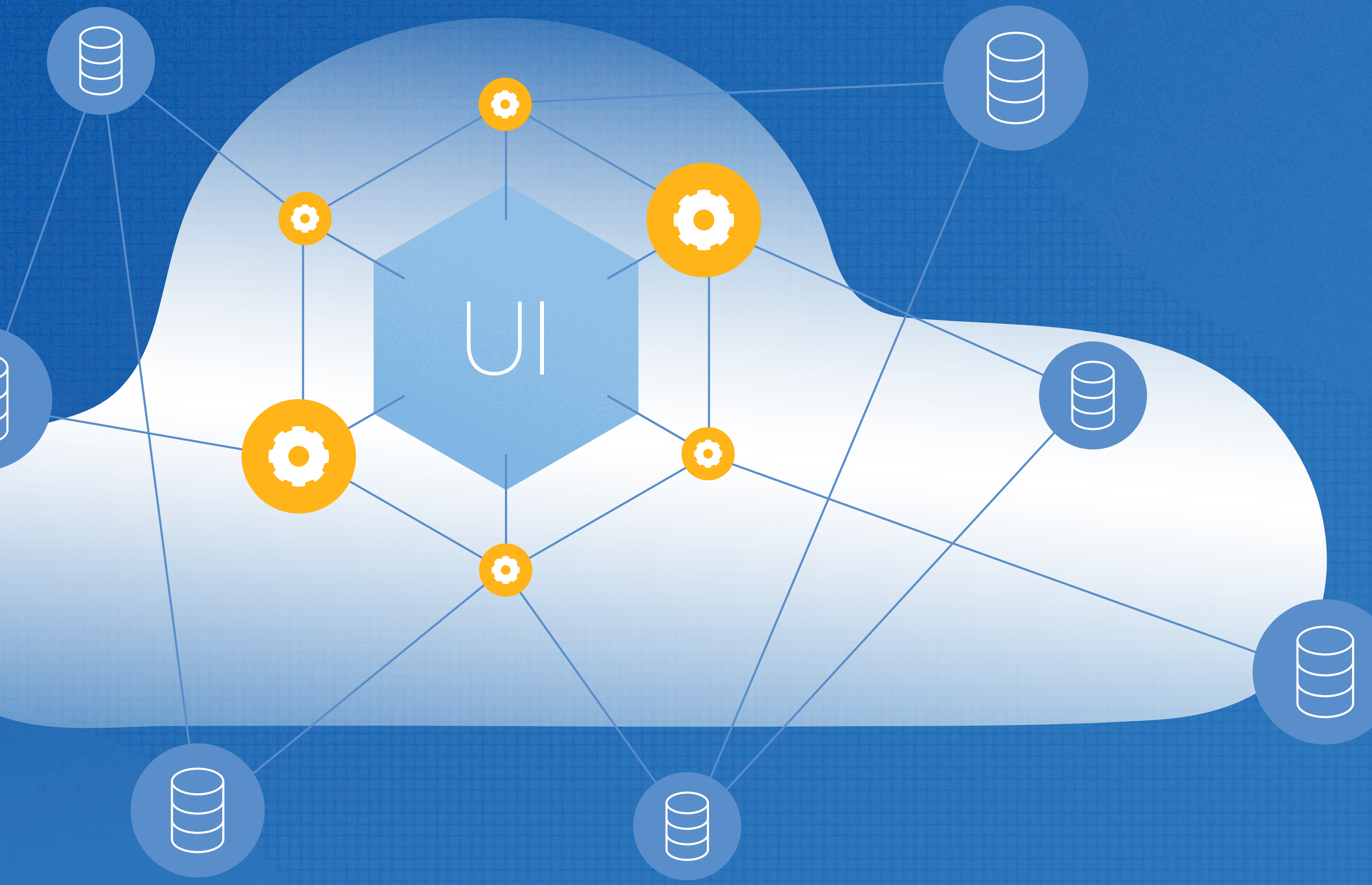
Progress®



Delivering Apps in Smaller Parts

Simple strategies to enable legacy apps to meet the needs of mobile users.

EBOOK

Introduction

Most companies run their business on software. There are exceptions, such as very small businesses, but in general if your company has more than one employee, there's at least one software application where the brains, or at least the memory, of the business resides.

Larger businesses have many more moving parts, so there's likely a lot more software running the business. This software, depending on the company, can be:

- One large app (can anyone say SAP?)
- Several large apps
- One or more large and small apps (could be standalone or wired together, who knows?)
- A bevvy of small apps (likely standalone, but could be cobbled together)
- Something between the other options (it's the wild west out there, I'm telling you)

If a large or medium-sized company is more than ten years old, chances are a mobile workforce or a mobile customer base wasn't even considered when building all these applications, especially if the apps are home-grown. If the apps are commercial offerings, then there may be mobile capabilities available from the vendor, but for large apps (or large companies) it's quite expensive and time consuming to bolt all those mobile modules to existing applications.

Fast-forward to today, and the needs of your company's workforce, partner community, and/or customer base have changed. They aren't chained to their desks anymore, they're out and about doing business. The purpose of this eBook is to propose a simple solution to enable your non-mobile legacy business apps to address the needs of these users.

Monolithic Apps to the Rescue

No, not so much.

Sophisticated businesses run on big monolithic apps because they must, not necessarily because they want to. When these businesses have:

- Multiple product lines
- Multiple divisions
- Complicated processes
- Deep partner or distributions channels
- Large workforces
- Large sales or support organizations

...the software that controls everything and keeps the business running smoothly is massive, complicated, and consists of several interconnected systems. Companies often refer to this class of software as Enterprise Resource Planning (ERP) software. Running large businesses with independent software systems is a nightmare, so that's why companies rely on as few systems as possible.

Adding mobile users to the mix changes everything.

If your company deployed an ERP system or core business apps that support mobile users as first-class citizens, then congratulations! You're done here.

Now, for the rest of you...

If you think about it, the mobile world we live in isn't really that old. Many people inaccurately believe the world became mobile with the launch of the iPhone in 2007 (which was only 12 years ago), but it started much earlier. Before that, businesses enabled mobile workers with BlackBerry devices or put cellular radios in laptops or personal digital assistants (PDAs) so employees could do business while away from the office. In most cases, doing business on the road simply meant having access to email while away from the office (BlackBerry) or using the same big, clunky ERP application on your laptop wherever you were, using the cellular radio to connect live, or sync with the home office, whenever you had radio coverage.

BlackBerry users were fairly happy with the results, as they could stay in touch no matter where they were. They weren't running a lot of business apps on their devices, but the world didn't fall apart when they were away from the office. They at least knew what was going on and could manage mail, calendar and tasks on their phone.

Laptop users were fairly happy as well, because they could at least get work done when out of the office although for many it was only when they had a solid cellular connection for their laptop. However, running your ERP system (or some other business software) on your laptop while on the road doesn't take into consideration the needs of the mobile worker.

Those solutions were a stopgap, hacks implemented by companies to enable mobile workers to do more when away from their desks, but it's not enough. Unless your business apps innately understand mobile users, you're not delivering the tools your employees, partners, and customers need to be successful while away from their desks.

Addressing Mobile Worker Needs

In general, the way an application user interacts with a business application varies depending on the tasks they're performing at the time and where they're performing the task. For example, processing monthly or quarterly reports is probably in-the-office work, and checking the status of an order or monthly sales number for a customer is need-to-know-something-now work.

You can refine if further and say that there's even a difference in the types of actions you'll perform in an application depending on what you're doing (or where you are) in the office (at-my-desk tasks or in-a-meeting tasks). In-a-meeting tasks are similar to on-the-road tasks, but likely only a subset. They're typically get-an-answer-to-something-now, or quickly-get-something-done tasks.

Now, that doesn't mean that an office worker doesn't look things up, or remote workers don't process stacks of paperwork. The point is that the needs of the mobile worker are different than the needs of a desktop user.

Developers building business apps for mobile users must rethink their approach. They can't just bolt a mobile UI to an existing system, especially those systems that have been around for a long time. Since the mobile users needs are different than the office user's needs, bolting a mobile-friendly UI on an existing desktop only makes the mobile user's experience slightly better and may not be enough to make users happy.

Few mobile users want or need every capability from the desktop version of the app; what they need is just the app functionality required when they're away from their desks. Mobile users have no time for complicated menu systems or long, multi-part forms. Instead, they want compressed workflows, smaller forms, even voice command or chatbots to enable quicker interactions and faster time to answers than what anybody can do with a monolithic app. They want one-tap access to the app function they need. They want streamlined (compressed) workflows that work the way they do—not the way the app, or the business, wants it. They want fast, quick and easy, not big and slow.

Targeting User Roles

Another factor affecting how a user interacts with a business application is the role a user has in the organization. Leaders, managers, sales, support, delivery, warehouse, etc. may all use the same apps, but they use them in completely different ways. Based on these roles, users may:

- Have different views of the data (including not being able to see some of the data)
- Have different actions they can perform on the data
- Have different rights to the data
- Use completely different parts of the app

This requirement further complicates the task of enabling your business apps for mobile users. Not only must you build the mobile friendly UI for the app, you must also clearly understand each user's role at runtime and apply all the required permissions and workflow steps in the UI.

If your app uses a web UI, this shouldn't be too challenging, since most of this role-based permissions stuff is already baked into the existing app UI. For more sophisticated or complicated apps, users will want a native mobile app, and delivering all that on multiple platforms (likely Android and iOS) will be a nightmare, especially when you add automated functional testing to the mix. More on this in the Progress whitepaper, The Cost of an Enterprise Mobile Strategy.

Supporting Workflows and Events

Mobile app users are…mobile. They're traveling from city to city, moving from customer to customer, store to store, job to job. They're not sitting at a desk, staring at an email client looking for work to be done. They're interrupt-driven . That means these users expect the business apps supporting (or is it driving?) them to tell them what needs to be done next.

Mobile users hop in and out of apps as needed during the day, because they're knee deep in workflows (sales, service, relationship management, partner development). They're not sitting in an app for hours, sifting through their work backlog. They're going in, figuring out what needs to be done next, doing it, and backing out again.

This approach mandates that business applications understand the business processes in play and deliver the quickest and simplest way to manage those mobile workflows for the user.

Inside those workflows are events, something that moves a business process from one workflow step to another. Events trigger automatically or manually:

- Time-based: such as contract renewals or scheduled maintenance
- Externally-triggered: parts or manufacturing raw materials arrive in a warehouse, customer PO arrives, etc.
- Internal: checks cut, purchase orders generated, etc.
- Manual: a workflow participant completes a task

What triggers an employee's participation in the next step in a workflow? Notifications.

Many enterprise apps don't do a great job at surfacing events and notifications to users. They may use email as a notification mechanism, but that forces users to work through their notification queue (their email inbox) to understand what needs to be done next. If the user doesn't prioritize their inbox correctly, they can miss important notifications. On top of that, email notifications sounds are binary (either on or off), so users can't get custom notifications for higher priority items.

Emails are best for communicating text. When used for sending notifications from systems, the included data is often inadequate and useless to act upon. For example, an expense report submission that generates an email notification including just the Report ID and a link to open the request in the expense

[1] https://www.encyclopedia.com/computing/dictionaries-thesaur uses-pictures-and-press-releases/interrupt-driven

system, does not provide enough context to enable the recipient to make an informed decision without context switching. That's why most system-generated emails end up migrating to a separate folder that we never open—making email notifications next to useless for mobile users.

If you're going to send me a notification, why not just send it directly to the application I'll use to dig into the details and act? Don't send me generic data expecting me to click over and work in the target app—send the notification directly to the target app. Backend applications can send notifications to devices that the device delivers directly to the user through the home screen in the notifications area or directly to target applications. Since most mobile app users limit the apps that can receive notifications, this guarantees that important business notifications get the user's attention.

Notifications sent directly to an app that displays an alert count on the application home screen icon alerts the user to important items in the app that require their attention. Users can customize notification alerts (vibration, sound played, etc.) per app, enabling them to quickly configure just the right notification settings for the apps that mean the most to them.

Addressing App Complexity for Mobile Users

As I've proven by now, monolithic enterprise apps have their place, but not on mobile phones. Parts of the app, but just not all of it. So how do you deliver just what mobile users need, when the need it, on their phones and tablets?

Microapps

One of the most challenging issues facing enterprise app developers is how to transform large screen app pages to the smartphone's smaller screens. I'm proposing skipping that entirely, and instead, delivering small, self-contained app parts to mobile users in a way that makes it easy for the user to just use the parts they need when they're mobile.

Each microapp encapsulates a single workflow or a specific part of a bigger business workflow, quickly getting a mobile user to the work they need to get done now. Modern enterprises build their catalog of microapps, hitting the important tasks mobile users must address while away from the office.

Each user/role will likely have multiple microapps they use, with some overlap across roles.

You probably won't let users deploy them willy-nilly on their devices, as that just takes too much work and will plunk a bevy of app icons on the user's mobile device home screen. You could use Mobile Enterprise Management (MEM) to manage deploying microapps to user devices, but you still have the problem of overloading users with too many app icons and grouping them together in a way that makes sense for the user.

The best approach is to deploy microapps in a microapp container, a single app that gives users quick access to all their deployed microapps. Within this app, microapps display as a list of items or as a grid of app icons. To make microapps and the microapp container work for them, users must be able to organize and arrange their microapps in a way that makes the most sense for their way of doing business. Just like you wouldn't force a common structure for desktop icons on a desktop computer, you can't do it for mobile users either.

To trigger business workflow action, use mobile device push notifications, instead of email, to deliver targeted notifications letting the user know when there's something they need to do in a microapp. These notifications go right the app, instead of the corporate email sinkhole, so the user sees them directly connected to the app where the work happens, to resolve items or keep workflows moving.

Notifications appear two different ways in the microapp container—as a badge count on the microapp container home screen icon, letting users know how many pending tasks await them inside the app. Next, inside the app, show the appropriate badge counts on all microapps so the user can see and prioritize their work in the apps that need their attention.

Earlier we discussed the relationship between workflows and events; workflows trigger events that end up as notifications in a worker's email inbox. Microapps short-circuit this, connecting the notification directly to the app where the user works.

The following table highlights some of the differences between big enterprise apps and microapps.

Monolithic Apps	Microapps
Everyone gets the same app	Only what the user needs, based on their role
Notifications show up in a different app (email)	Notifications show up in the app I use to resolve them
Can't deep link from an email notification to the work I need to do in the app	Opening the app gets me right to the work I need to do
Online only	Can easily be offline enabled
Tailored for no one	Tailored for me, or at least for my role
Navigate to where you need to be	Jump right in
App has features you'll never use	Apps have only the features you'll use
Multiple logins for different apps	Single login for the container app
Disparate user experience, each app looks and acts very differently	Unified user experience, feels like you are using just one app

Don't Throw Away the Old

Microapps are an extension of your existing enterprise apps; you won't get rid of any existing infrastructure and many of your desktop users will continue to use the apps as they do today. Microapps are simply small, targeted sub-apps targeting specific workflows or specific parts of larger workflows.

Don't be confused—there are a lot of microservices articles circulating that show how to split big tasks into smaller ones. What we're talking about here is the latest innovation in that journey for mobile apps. The apps use your existing app infrastructure (you can only have one system of record, right?), so existing users aren't impacted, only mobile users get a better, more streamlined experience for the same app.

Your existing app should expose the services (web services, most likely RESTful services) your microapps consume to work within the framework of your existing business app. If these services don't exist, or you can't access them from "outside" the app, then you'll have some work to do to build the services you need. If the services exist, but they're not very mobile-friendly (for example if they're SOAP-based or send more data over the network than is needed for the microapp's needs), then you're going to need your own services that piggyback on the existing services exposed by your legacy app.

Just the Data You Need!

As mobile devices became more powerful and mobile networks pumped up the available bandwidth, developers stopped caring about the impact their apps had on the device and network. Want to know why your device battery won't last a full day with heavy usage? The quality of mobile applications.

Mobile apps regularly get more data than they need from the servers and must discard data that should never have been sent their way. This means additional charges for app users who have data limits on their accounts or live in areas with limited bandwidth availability. They also get unprocessed data, forcing the app to perform calculations or data transformations that should have been done on the server before sending the data. This means increased load on the device processor, which impacts battery life.

Make a sign with the following and post it near your development teams as a reminder:

Rules for Mobile Developers:

1. Never send data to a mobile device that the device doesn't need.
2. Minimize the amount of work a mobile app must do to render its data.
3. Look for ways to reduce your app's impact on battery life, which means reducing processor and network utilization.

Delivering on this requires smarter app architectures, and, honestly, just paying attention to everything you put into your app. Look at every feature, no matter how small, with an eye to the feature's impact on processor utilization, memory, and network bandwidth consumption. Wherever possible, cut things back until you can deliver on the feature's requirements with the minimal impact on the device battery and network utilization.

Building Smaller Apps

Okay, so how do you build microapps? Delivering these apps requires an intense understanding of how users use your existing legacy apps, especially when these users are outside of the office on mobile devices. This involves:

- Identifying the key mobile roles using the app
- Surveying a representative sample of users in each role to gain a better understanding of their needs
- Doing ride-alongs to watch key users at work, or looking over their shoulders as they work from their desks
- Build workflow diagrams highlighting discrete tasks users perform in the app
- Review the workflow diagrams with a representative sampling of users in each targeted role
- Stack rank workflow and role combinations so you know where to have the biggest impact before you start building anything

Once you know how these apps are used by mobile workers, start designing the app UIs for microapps addressing the most common workflows:

- Break legacy app flows and break them up into smaller parts
- Don't try to deliver anything more than the minimum required for the task at hand
- Look for ways to use voice control or bots to streamline workflows; there's no need for an app when answering a direct question from the user is enough
- Don't worry about overlapping microapps (two apps that do almost the same thing); if the user role requirements are slightly different, you'll break things (and create unhappy users) if you try to stuff two role/workflows through the same microapp.
- Understanding which workflows are time-critical, and build the appropriate notification strategy for those tasks.

Deliver just the parts of the app the user needs, nothing more, nothing less. You can never have too many microapps, provided the microapps properly address the needs of mobile workers and enable them to get more done faster.

Deliver just the right notifications as they happen and display them as simple actionable forms. Be sure to build in an accommodation for snoozable alerts, supporting tasks that are time-important, something that needs doing soon, but not right away. You'll create much happier users when your notification strategy understands task importance with more granularity.

With all this preparation work done:

- Select a platform that enables you to efficiently manage delivery and management of microapps on mobile devices. Ensure that the deployment and management framework adjusts its output based on user roles so users only get what they need on their mobile devices.

- Build out the services your microapps need, keeping in mind the developer rules outlined in the earlier sidebar

- Start building and deploying your microapps, targeting highest impact areas first (looking for the highest productivity boost, or biggest impact on employee satisfaction)

Be sure you have a system in place to measure the success of this work; there's no reason to do any of it if you can't prove success or at least identify failures. Available options include:

- Instrument microapps so you can capture when and how the workers use microapps. Pay close attention to the paths users take through each microapp and abandoned workflows, this creates a blueprint for the app's next refresh.

- Survey users periodically (monthly, quarterly, etc.) looking for ways to improve the quality of the delivered solution.

- Look for ways to measure pre- and post-deployment productivity for affected users. Use the data you collect here to guide future work and budget.

- Provide a quick and easy feedback mechanism in your microapp or microapp container; when users know you're listening, they'll give you an earful of useful information.

Rework failed microapps or just kill them. If users won't use one or more microapps or aren't successful using them, what's the point of having them?

Listen to your users; you're delivering microapps to make mobile workers more successful or efficient while away from the office. If it's not working, then you must rethink your strategy. Are there not enough apps? Too many? Are the apps too complicated? Too simple? Why isn't this working?

Beyond the Core Capabilities

In this article, we described how to build and deploy a suite of microapps designed to create more efficient mobile workers. We talked about clearly understanding user roles and delivering the apps designed to accommodate users in each role.

Flexible Deployment Options

Be prepared to accommodate outliers who don't fit pre-conceived role models. In some organizations, it may not make sense to view roles so rigidly. An option for these organizations is to publish a microapp catalog and let users self-provision the apps they need into the microapp container described earlier in this paper. This lets users pick the 'apps' they need and is especially important for workers who may have more than one role.

Tailoring Notifications

When building and deploying microapp solutions, be sure to make notifications configurable. Users will have different pain thresholds, that point where notifications stop being helpful and become irritating instead. Let users decide what they want pinged on and what they don't.

If possible, build your solution with an understanding of employee vacations through integration with enterprise calendaring systems. Let users configure, at a global level, different notification strategies depending on whether they're marked out of the office (and not working). For some users, that means no notifications while on vacation, while for others this means only urgent notifications while on vacation. One set of rules will not accommodate all use cases.

Also consider using that same calendar integration to mute all notifications during meetings. Queue those notifications up and deliver them only when the calendar shows the user isn't otherwise occupied.

Notification Hierarchy

For some roles, or for certain customer issues, a simple notification to an on-device app won't be enough. For business or customer-critical events, consider enabling users to configure notifications using SMS or even phone calls. This approach helps ensure that important notifications are seen in a timely manner.

Synchronize Across Devices

Survey your employees and you'll quickly see that most carry more than one mobile device. With that in mind, your microapp container should synchronize its configuration across all devices associated with each user. This eliminates the requirement to configure the user's microapp layout multiple times.

The same holds true for notifications. Consider more flexible options for notifications in multi-device environments:

- Notifications sent to devices sequentially (primary device first, then notify the secondary device of the user doesn't acknowledge the notification within a specific timeframe (configurable, of course).
- Notifications sent to all devices
- Notifications sent to all devices and dismissed on all other devices when acknowledged on a device

Streamlining Microapps

Disorganized development organizations have an issue with islands of code. Smart organizations move to blocks of functionality.

Let me explain.

To gain efficiency (do less work), developers borrow software components, algorithms and other stuff from different parts of the organization. If you have an innersource program in place, you at least have a common place to share code across the organization. If you don't, your developers are copying and pasting (C&P) code wherever they can get it.

C&P reuse is easy to do, and it has the added benefit that there's limited future impact for these developers - they copy the code, test it in place, and, if it works today, it will likely continue to work (pending any dependent system changes).

Each app has its own copy of the code, and it's maintained in place. If someone finds a bug or a security vulnerability in the code, they just fix it and go on their way. Unfortunately, that means that each team fixes their version of the copied code—but only if they discover the issue before there's a costly failure or an embarrassing data breach.

That's not the right way to run your business systems.

Development organizations that recognize the C&P approach in their development work streams quickly work to find a better way. This leads to building shared libraries and a mandate to use them in all corporate apps. The library approach is a good one; the code's written once, tested on a variety of systems, bugs and feature requests filter back, and fixes and new features eventually make it into the library. Building libraries requires more work as API signatures and system architectures must be carefully thought out before coding even starts. But delivering building blocks your developers can use to build their apps is a much better approach than C&P. Some considerations for this approach:

- Shared libraries may make it into the new app, especially if there's a company mandate, but older apps rarely get fixed. This means that old, flaky code is still running in your customer-critical applications and you don't have the budget to fix it.

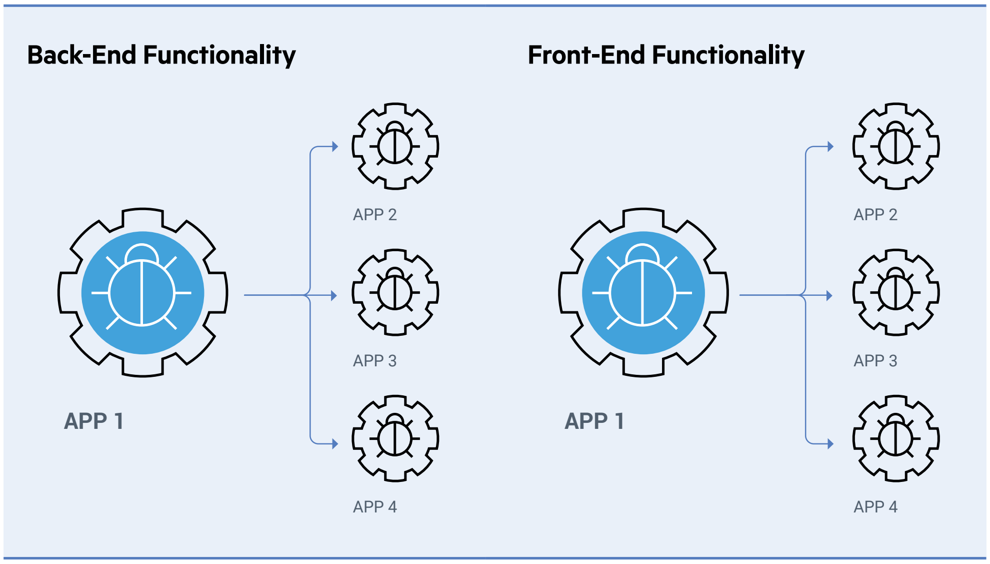
- Systems built using incompatible development languages or platforms force organizations to port libraries for these other targets. This dramatically increases the cost of development, testing, and support for these libraries plus extends release schedules to the point where some apps have the latest, best libraries, but other apps are waiting for their version to go live.

Some organizations try to address the organization's needs through a solid and well-thought out API architecture. The thought being that the special sauce is in the APIs and having an enterprise-wide catalog of APIs means everyone can use them and maintenance and support resides in one place. Each API has a specific purpose and is shared with one or more applications. With this in place, apps reuse the data connections, algorithms, and other business logic encapsulated in the API layer leaving more time to spend on the app UI and user experience.

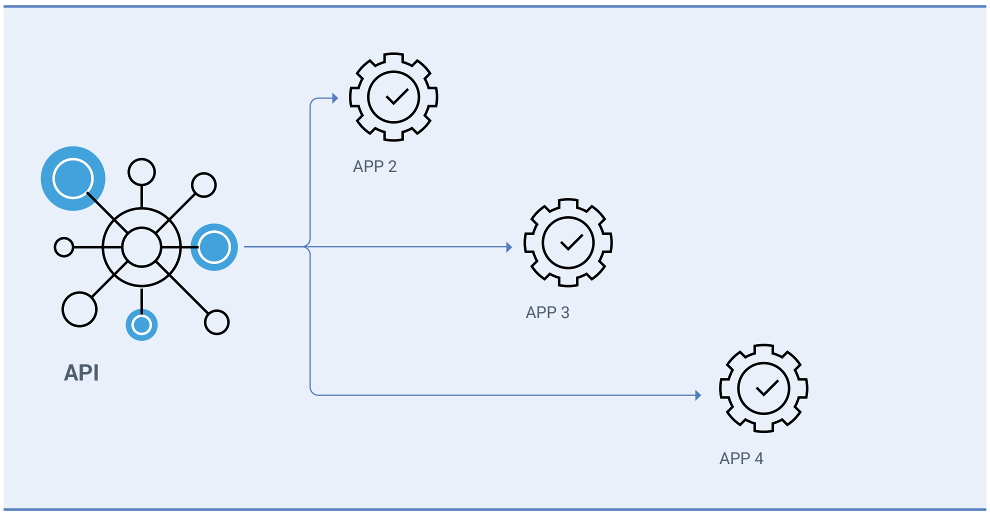
APIs are backend architecture, apps must implement their own front-end or client-side code to process the information returned from APIs. So, while APIs are very useful to share backend functionality, they do nothing to deliver code reuse for front-end development, skipping all together user interface elements, screens, workflows, and data validation.

A more robust approach to code reuse is Functional Sharing. Rather than share one or more parts of a component implementation, with functional sharing, you instead reuse the implementation of user goals, like “Search for a Customer”. With this approach, share both frontend and backend components for maximum benefit and maximum reuse. Functional sharing works in a microapp architecture because each microapp contains everything it needs to complete the user goal. Further, updates to the functionality are delivered to all users, skipping the work needed to port the new feature in other apps. When you need functionality from multiple microapps, wire them together, or wire the internal parts of them together, to deliver something bigger than the sum of its parts. Since microapps encapsulate a single action or transaction, updates are much easier to test and deliver. This encapsulation ensures you’ll have less dependencies to manage!

COPY & PASTE



APIs



Reuse at the Functional Level

To gain efficiency, application developers have approached sharing software elements and processes in many ways. have shared functionality in different ways. Each approach brings both benefits and tradeoffs.

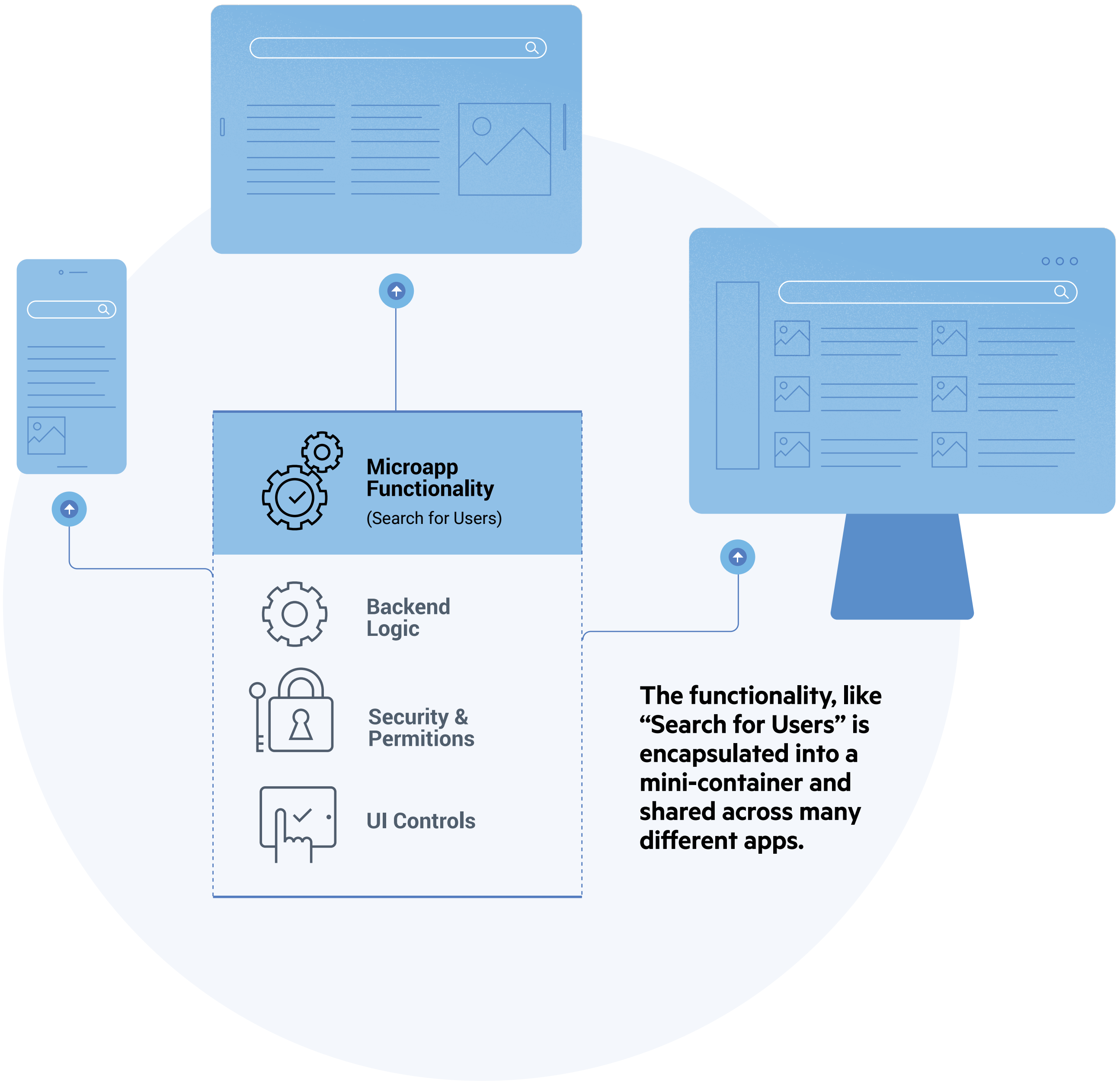
Copy+Paste reuse is the easiest to do and has the least amount of future dependencies. Each app gets a copy of the source code being shared. Since the apps aren't using the same piece of code this code will quickly get out of sync, spreading bugs everywhere. A discovered bug must be fixed in all the places using that source code. Additionally, any alteration to the code, whether a new feature or some other adjustment must also be implemented in all the places using a copy of the shared source code. Most developers will copy and paste code, but it's not a recommended approach for any serious architecture. Both front-end and back-end code can be reused with copy and paste.

Another reuse approach is to package functionality into a library and share the library with other applications. Most applications make use of one or more libraries. This approach is encapsulated and makes library updates easier to deliver. At the same time, building a library for sharing requires more design and architecture effort than copy/paste. It is almost always the case the programming language of the app must match the programming language of the library. In polyglot departments using more than 1 programming languages, there is always the chance functionality will need to be reimplemented in another programming language causing inconsistencies as the applications diverge over time. Generally, back-end source code is packaged into libraries though this is not always the case.

A third option for reuse that is very popular is to use an API architecture. Applications connect to APIs for data and processing. Each API has a specific purpose and is shared with one or more applications. Apps reuse the data connections, algorithms, and other business logic encapsulated in the API layer. APIs are considered a back-end architecture, the apps must implement their own front-end or client-side processing to work with the information returned from the API. So while APIs are very useful to share back-end functionality they do nothing to help reuse of front-end development like user interface elements, screens, workflows, and validation.

Functional sharing is a different approach to reuse. Rather than share parts of an implementation, Functional sharing means reusing the implementation of a user goal, like "Search for a Customer". Both front and back end components are shared for the maximum reuse. Functional sharing works in a micro app architecture because each micro app contains everything it needs to complete the user goal.

Further, updates to the functionality are delivered to all users, skipping the work needed to port the new feature in other apps. Micro apps can be implemented with other languages and tie in to different data sources. Since they are encapsulated, updates are much easier to test. They have a lot less dependencies to manage!



Microapp Functionality
(Search for Users)
Backend Logic
Security & Permitions
UI Controls
The functionality, like "Search for Users" is encapsulated into a mini-container and shared across many different apps.

About the Author

Dan Wilson

Dan Wilson is the Senior Product Marketing Manager for Mobility technology at Progress. Dan has extensive experience growing technology focused products and services. He got his first taste of fast-moving bleeding edge tech when he joined his first start-up in 1999. An avid participant in technology communities, he contributes to a variety of open-source projects and presents at numerous developer conferences worldwide. Prior to joining Progress, Dan founded and directed a consulting practice for 10 years.

Find Out if Microapps Are For You—Progress Kinvey Can Help

We'll help you understand how microapp technology can help you deliver more mobile functionality to your users. Our experts can walk you through the concepts and discuss your infrastructure and current application catalog. Contact a Kinvey expert today.

Talk with an Expert