Accelerate Mobile Engineering

With Standardized Tooling

Embrace

Introduction

Whether you've come into an organization to manage an existing mobile team or to create a new one, you're taking responsibility for several critical objectives in parallel. This includes maximizing resources; maintaining a high velocity of product releases; and improving the overall performance and satisfaction of your mobile team and individual developers.

That is a lot of responsibility at any organization, especially in the mobile space. The challenges of mobile are different from other development teams. We understand those challenges.

Separate teams have to build a unified user experience on different code bases with quick turnover between already short release cycles. To succeed, mobile teams must quickly, continuously, and efficiently produce quality software.

To do this, we should keep in mind the principles of **velocity engineering**: an organization-wide framework for making cultural and technological choices to improve developer productivity, efficiency, and happiness.

We can teach you how to use velocity engineering to reduce complexities and simplify the environments and tooling your mobile teams rely on. We're experts at this. Every day, we help hundreds of teams like yours overcome these challenges as their organizations grow and mature.

In this eBook, we'll show you how to:

- **Realize massive cost savings**
- **Reduce complexity**
- **Increase efficiency**

By implementing these best practices, you'll increase team satisfaction while maximizing efficiency. Happier engineers deliver better products, faster.

Mobile Is Different and Challenging

Mobile is new and requires specialized tools and a different way of working. Organizations new and old are learning for the first time the growing pains of managing mobile-only teams.

These are some of the most common challenges we hear from our partners:

- Mobile teams have separate tooling, so collaboration is more difficult.
- Mobile teams with separate tooling don't necessarily have access to the same data.
- Product managers find it difficult to track feature development across mobile teams.
- Product managers try to implement web features directly into mobile.
- Product managers try to use web tools for mobile projects.

What is it about mobile in particular that make these challenges so common?

Your company's product managers likely have a long familiarity with web development, which is far more commoditized — and therefore makes it easier to outsource work and monitor features. In contrast, your mobile teams are using more modern and varied stacks, whether native to iOS and Android or using a cross-platform framework through React Native or Flutter.

And the process for releasing mobile products is more complex than for web. It's like a black box, with multiple ecosystems, operating systems, and rules. This makes tracking features, releases, and related issues much more difficult.

Thus, it's not possible to simply port web features to mobile apps. A common frustration is the expectation that features should be rapidly and easily ported to mobile. For example, a company with a successful web notification system may want to quickly deploy on mobile. Here, they must recognize that the mobile ecosystem mandates they build new — and separate — notification systems for iOS and Android.

Additionally, not all web performance metrics, such as page rendering time, have an immediate analog on mobile.

Save Costs, Reduce Complexity and Increase Efficiency

Next, we'll show you how to standardize tooling and software development kits (SDKs) to realize the following benefits:

Massive Cost Savings

- Reduce time to release by focusing on product development.
- Save money on license costs by eliminating duplicated software.

Reduced Complexity

- Maintain the same tooling for each mobile team to reduce communication errors.
- Eliminate unknowns and tooling incompatibilities.

Increased Efficiency

- Solve bugs quickly and centrally.
- Consolidate processes and solution workflows across all teams.
- Onboard developers easily to new teams or apps.

The bottom line: You'll have happier teams delivering better products, faster.

It's critical to use your mobile developers' time efficiently. Mobile developers cost 2-3x more on average than other developers. By freeing them from managing unneeded tools and making it easy to onboard to new teams or apps when needed, you will realize massive productivity gains in what you actually hired them to do!

Why Embrace? Mobile is unique. All organizations adding mobile are facing the same challenges, and no teams have experienced these challenges before. If you can't look internally to teams who have been there and solved it before, who can you look to for solutions? Every single day we talk to organizations with mobile teams who face the same challenges you do. These companies are at many stages: just introducing mobile, mobile-only, one mobile team, or many. We are in close daily communication with our partners. We know their pain points intimately and we're in the trenches with them, helping them grow and mature over time. We know the communication gaps between iOS and Android teams and between mobile developers and product managers. We know how to overcome these challenges. This is why we exist and why we've created this blueprint for you.

The Velocity Audit

As more of your organization's critical internal and customer-facing applications shift to mobile, it's vital to understand the processes and tools your team uses so you can measure inefficiencies and remove duplicate tools. In this audit, we'll show you how to identify processes that may be creating bottlenecks and replace them with best practices geared toward increasing velocity on product releases.

> ***What Do We Mean When We Say "SDK"?***
> *We use this term a lot so we want to make sure we're all on the same page with a common definition. We think about SDKs as "services in a box," or as the interfaces to vendors' products. These enable you to connect your application to those services or products quickly and easily, reducing the time and resources required by your team. At app load time, the service or product begins running quickly and automatically in the app.*

The Velocity Audit

Step 1: Internal and External App Audit

Step 2: SDK Audit

Step 3: Identify Outliers

Remember, the goal here is to reduce costs, complexities, and inefficiencies. It's not to cast blame or point fingers at individuals, teams, or traditions. Bringing your team along with you and highlighting the benefits is important. When we work with partners to conduct these audits, we make sure teams recognize that increasing release velocity is both a cultural and technological process. When your team begins to understand the rationale behind this approach and see the measurable results of acceleration, everyone wins.

Step 1: Internal and External App Audit

This step is particularly illuminating if you are just coming into an organization to manage a mobile team. We have worked with partners who realize after several months that there are "dinosaur" or legacy apps still utilizing resources — human, time, tooling, and otherwise — that weren't immediately visible. It's critical to have a clear map of all internal and customer-facing apps your team is responsible for.

In this step:

- Identify all (and we do mean all!) of your internal and external apps. Determine which resources those apps utilize (infrastructure, network, SDKs, etc.).
- Document where the app components "live" (code base, infrastructure, etc.).
- Identify which team members work on which apps, and where there is overlap between them.

The Bottom Line: Get as complete a picture as possible. For every app, understand what tools are used, what they're used for, and who manages them. The more information you have, the easier this process will be.

Step 2: SDK Audit

Now that you know which apps you're responsible for, you need to understand how those apps are being managed and monitored, by whom, and with what tools. There are likely many tools in place — real-time analytics, logging, crash reporting, alerting, infrastructure monitoring, product analytics, attribution, real-time bidding or ad servers — depending on the type of app and user base. If each of your apps has duplicated SDKs or if multiple team members handle the monitoring and management of the SDKs, you are accruing significant tech debt and inefficiencies. It's also important in this step to identify if multiple teams are paying for the same tool. Duplicated software licenses can really increase expenses.

The goal of this step is also to minimize complexity. Each SDK adds complexity in the app. The more SDKs we can prune or consolidate, the better.

Why is it important to make absolutely sure you also document the free or home-grown tools your teams are using? The saying "there's no free lunch" is applicable here! There's a cost to "free" or home-grown SDKs. Your teams are racking up engineering costs, time costs, and the potential costs of a lack of standardization across apps if you aren't aware of these SDKs and can't assess whether to implement this tool (or another one) across all apps. It's critical you know about ALL tools.

In this step:

- Document each tool being used for each app.
- Document what each tool is being used for.
- Check for SDKs that aren't being used. Some SDKs are installed in apps and are never called in the code. We've seen this happen many times.
- Make sure teams tell you about the free tools they are using. (It's easy to forget about tools you don't pay for!)
- Make sure teams tell you about SDKs or tools they've built themselves. (It's easy to forget about home-grown tools that have slowly become part of the landscape over time, and may not even have a name.)
- Document who manages and monitors each SDK for each app.

The Bottom Line: It's absolutely critical you know exactly which SDKs are being used for each app, and for what. Without this insight, you can't reduce costs, complexities, or inefficiencies.

Step 3: Identify Outliers

In the previous step, you identified which SDKs are being used for each app and for what purpose. In this step, you need to determine if there are teams using a completely different SDK than most of their peers, or if they are using an SDK for a different function than everyone else. The goal of this step is to identify — and minimize — outliers so you can standardize as many tools as possible, to increase efficiency and velocity.

In this step:

- Identify which SDKs are being used for which functions, grouped by teams/apps.
- Identify if any teams are using a totally different SDK for a specific function.

The Bottom Line: Be aware of what tools are being used for what, for each app, and if there are any outliers. This will help you standardize.

Understand how your teams are using SDKs for specific functionalities. For example, some of your teams may be using LOG4J or Lumberjack. Some may be using a custom-written tool, while others may be logging by hand. And others may not be using any logging tools at all. Analyzing the SDK landscape app-by-app will help you identify if there's one app or one SDK that is an outlier. (We'll explore what to do with outliers in the section on best practices.)

Now What? Best Practices for Velocity

Just as it is important to bring your team along collaboratively during the audit, we encourage you to include your team in implementing these best practices. When everyone is empowered to participate in the decision making and execution of a strategic plan, they feel ownership of the outcome.

Best Practice 1: Standardization

Best Practice 2: DevTools

Best Practice 3: Self-Service

Best Practice 4: Baselining

Congratulations on completing the velocity audit! Once you've gathered the key information in the three steps above, you have what you need to reduce costs, complexities, and inefficiencies so you can optimize your release velocity. The decisions you make from this point forward will be impacted by several factors, including the size and structure of your organization, your budget, technology resources, and your team culture.

Best Practice 1: Standardization

In Steps 1 and 2, you identified the SDKs your teams use across your internal and external apps and documented what they used each tool for. Now, your teams can decide which SDKs to unify and standardize on. This also makes it easier for your remote or distributed teams to access and use SDKs across different network configurations and environments.

Is There a "Golden Number" of SDKs?

We get this question a lot: should my mobile team target an ideal number of SDKs to support our apps? The answer varies, depending on your organization's domain and the functionality of your app(s). The goal for simplifying and unifying your tooling environment is to reduce the total number of SDKs your team manages. This also reduces the size of downloads, increases start time performance, and reduces the number of bugs you need to troubleshoot. We aim for ~10 SDKs for an "average" app. If your app supports real-time advertising and bidding, for example, it will of course require more SDKs and tooling.

With everyone logging into the same tools and accessing standardized data sets, you can achieve an economy of scale. For paid tools, you may enjoy license and cost benefits. Team members will have fewer SDKs to learn and can quickly onboard to other apps within your organization without having to learn a new tool.

As you begin to standardize SDKs across your apps, you will likely be asked to make exceptions for certain functions. As a general rule, we encourage you to standardize

as much as possible. Outliers should be considered and approved when there's a specific purpose. These "legitimate outliers" should be kept to a minimum, for ease of maintenance and to maximize efficiency.

In some cases, it makes sense for a specific app or team to use a different tool, depending on how the app is used, or how critical it is to the bottom line. eCommerce organizations may have internal or external education or training teams that use more video content than teams working on the actual eCommerce apps. These radically different app requirements would lend themselves to a different toolset. Real-goods companies like Amazon have critical customer-facing apps that require the most-current tooling and support. Their internal apps (e.g. logistics) may be supported by a different set of SDKs that cost the companies less to maintain, as uptime doesn't require quite as many "9"s, and response time can be slightly slower.

Measuring Velocity

As you tighten your best practices and relieve the pressures on your developers by increasing efficiencies and abstracting SDK management away from development, you can begin to measure engineering velocity. For example, ask your teams to estimate time for routine tasks like cutting a release or documentation, and then have them measure actual time. Track estimation against actual time in a bug tracker or Jira, so that each task has an engineering hours "estimate" and "actual." After a few months, your team can start to predict actual velocity. This process can be used to create plans for the next quarter. Velocity increase can be achieved by continually helping your team reduce distractions, finding and reducing inefficiencies, and tightening estimations.

Best Practice 2: DevTools

The goal of this blueprint is to increase the overall efficiency and velocity of your mobile team. To do that, you want your developers to spend as much time as possible working on features and innovation. Time spent managing tools, monitoring dashboards, or analyzing crash reports is time your developers will never get back. Depending on the size of your organization, there are several ways to abstract the monitoring and management of SDKs away from actual application development. Earlier stage or smaller companies without the budget for a dedicated DevTools team can designate one or two engineers to make the decisions on which SDKs to standardize on. This forms the beginnings of a DevTools team as you grow and establishes best practices for abstracting tooling away from development. Larger organizations with budget can establish a dedicated DevTools team. These DevTools practitioners assume responsibility for monitoring and management of SDKs. Freeing up your developers to focus on their core jobs helps increase velocity more generally.

When you move towards this model, you reduce the hidden costs that each application team was repeatedly paying by supporting the same set — or, before standardization, a different set — of SDKs. For example, the release of iOS 14 caused issues for many applications. Without a DevTools approach, each individual applications group had to

figure out the replicated problems at the team level. For a large organization with multiple internal and external apps, that would accrue to a huge accumulated cost as each team unknowingly worked to identify an identical problem. With a DevTools team, the iOS problem would be identified — and resolved — centrally for all applications, saving time and resources across the organization.

Best Practice 3: Self-Service Access

Standardizing on SDKs and unifying your toolset enables you to give access to other stakeholders. This saves time and resources on issues that your mobile developers would otherwise be handling. For example, you can give QA or customer service (CS) access to SDK dashboards, so they can triage certain issues before involving your developers. With the right data, CS can determine that some app access issues are actually caused by network outages and won't need to escalate them to your mobile team at all. This will reduce overall issue volume, and will more precisely identify issues, ensuring the appropriate teams are responding to them.

Best Practice 4: Baselining

As you start the journey towards velocity, there is no better time to begin baselining. Whatever you can measure is helpful to prove your progress. You'll quickly see that as you adopt the best practices in this blueprint, you will rapidly gain velocity as an organization. Capturing metrics like mean time to repair (MTTR), time to deploy (TTD), number of releases, and average number of deployments will help you understand where you are efficient and where you can continue to improve.

Self-Service Examples

- ***CS Teams:*** *Triage issues from a health dashboard before contacting engineering.*
- ***QA Teams:*** *Check server deployment status when feature testing.*
- ***Product Teams:*** *Check success/failure of features from user experience standpoint.*

Managing the Human Side of a Velocity Audit

We've talked about the team-level benefits of consolidating tools and improving the overall efficiency of your mobile organization. What about the benefits to your individual developers? A well-functioning mobile organization is made up of happy developers who can focus on what they do best, and who aren't bogged down by managing multiple tools. If you encounter developers who seem resistant to this optimization process, you have a great opportunity to explore what's behind the resistance. Does that individual enjoy managing tooling more than development? They may be a great addition to a DevTools team. Or, their preference might make them a strong fit to help you with your quest for standardization. Adjusting to new tools they choose will help them realize the benefits of being free from tooling complexities. Help them understand and appreciate the increased time they'll have to drive innovation and company revenue by doing what they do best.

Conclusion: Start Simple and Accelerate

This blueprint has hopefully helped you take actionable steps toward realizing cost savings, reducing complexity, and increasing efficiency so your mobile team can focus on releasing better products faster. It can feel overwhelming at first if you've just come into a new organization with many legacy tools or disparate teams. The best way to stay focused is to start with the easiest areas: unifying your SDKs and enabling your QA and CS teams to leverage SDK dashboards and data. The immediate results will be a reduction in the number of issues across the organization and a noticeable increase in velocity and team satisfaction.

As velocity continues to increase, you can start to focus on other innovations, like more advanced baselining and release automation.

For more information or for help with your organization's SDK audit, please contact us at **truenorth@embrace.io** or visit our **website** to learn more about how to improve your mobile processes.