



Modern Product Management

Part 2: Execution

A TAPLYTICS GUIDEBOOK



Modern Product Management

Part 2 - Execution

In part 2 of the Modern Product Management series, we'll be discussing executing a high performance A/B testing and feature management program. The first half covers strategies and best practices for developing web and mobile experiments and the second half is dedicated to shipping faster, reducing risk, and enabling more successful product launches with feature flags.

Table Of Contents

- 1 A/B Testing: A Modern Product Management Requirement
- 2 Introducing No-Code Experimentation To Increase Speed
- 3 No-Code A/B Testing For Mobile Apps
- 4 No-Code Web A/B Testing
- 5 FAQ: Client vs Server-Side Testing
- 6 The Modern Product Manager's Guide To Continuous Deployment With Feature Flags
- 7 Why Phased Rollouts Are The New Standard



A/B Testing: A Modern Product Management Requirement

A/B testing (also known as split testing) is the practice presenting a sampling of users with two versions of a screen or experience (often a current “A” baseline version vs. a “B” variation with some visible change) and tracking user interactions with each version to determine if one more positively influences user behavior or engagement.


You’re responsible for boosting metrics like retention, conversion, and engagement, but also for creating a personalized experience for customers and collecting feedback from them.

There shouldn’t have to be a tradeoff between customers and products as you decide where to spend your time. The way to combat this dilemma is with experimentation. Running hands-off tests to find out what delights your customers helps you collect candid user feedback without consuming too much of your time. We’re here to walk you through the world of mobile experimentation and how it could make a world of a difference in your app. In this guide, we’ll explore:

- 1 Why experimentation?**
- 2 Where and what to test in your app**
- 3 Examples from industry leaders**
- 4 How to create an experimentation culture**

Why Experimentation?

The downfall of companies with traditional mindsets and business models like Blockbuster and Sears serve as a wake-up call to remind us all that stagnation has



never been the key to success. Customers are unforgiving to companies that are afraid to push beyond their comfort zone and ignore the value of experimentation. This is especially true as mobile becomes such an important part of modern business and customer experiences.

It's hard to predict the future, especially with customer behavior changing so rapidly. This is where adopting experimentation comes into play. Experimentation allows companies to listen to their consumers in real-time and iterate accordingly. To drive engagement, ROI, retention, and any other metric, you need to create with empathy and let your customers show you what they want!

Why An Experimentation Culture Is Valuable

- You're more likely to discover small, effective product changes.
- Hierarchy doesn't dictate product decision-making. Ideas aren't prioritized by position — instead, the validity of everyone's ideas is fairly measured through test results.
- Your team members feel motivated. According to a Forbes study, employees are more likely to be engaged in their jobs when they are more involved in projects.

Where & What To Test In Your Funnel

For each of your key metrics, there are several levers in your product you can experiment with to find out what moves the needle. Find out what areas you're doing well in and where you can make improvements to increase retention, engagement, and conversion.

Retention

Retention is the number of users who return within a certain timeframe - most apps use 3 months (90 days) as the benchmark. Being able to keep users coming back to

your app and using it more than once can be difficult - studies show that 21% of users abandon an app after only one use.

Onboarding

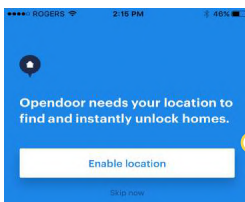
The best time to create active users is during onboarding; effective onboarding has been proven to increase user lifetime value by up to 500%. However, your users will only stick around if they find value in your product during their first experience. This is why it's imperative to focus on improving your user onboarding and continuously experiment with new ways to bring your users that value quickly.

Try experimenting with:

- Options for quick registration with social sign in
- Product tours that highlight your unique value-add features
- Notification opt-in screens and messaging

Push Notifications

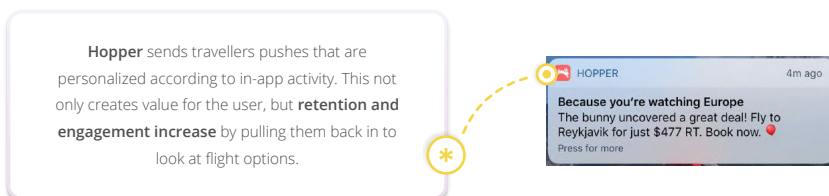
If they've opted into push notifications, users open an app 88% more than users who have not opted in. If they aren't reminded of what value your app provides, they won't be engaged and are more likely to churn.



Getting opt in for notifications and, depending on your app, location services, is crucial to retention and re-engagement. Try **providing specific context of what value they will receive** when you ask them to opt in or the option to opt in later like **Opendoor** does.

There are many ways that you can experiment with the use of push to increase app launches:

- What time you send notifications
- Sending images in notifications
- Messaging and personalization



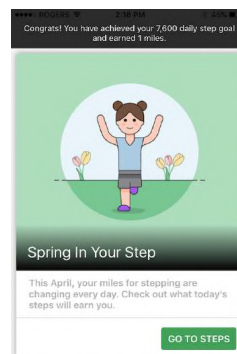
Create Incentives

Use offers and incentives to bring users back and keep them visiting. Companies often give up on users who aren't visiting anymore. Of potentially churned users, 30% would return if offered a discount, and 24% would return if offered exclusive or bonus content.

Such incentives could include:

- Unlocking new features
- Earning loyalty points or discounts
- Flash sales or offers

A great example of this is Carrot. Carrot lets users connect loyalty programs, which are attributed when the user completes personal fitness challenges and goal progression. They release new challenges every few days (via push!) to bring users back into the app.



The user is incentivized by more free points that are useful and easy to earn by completing challenges.

Engagement

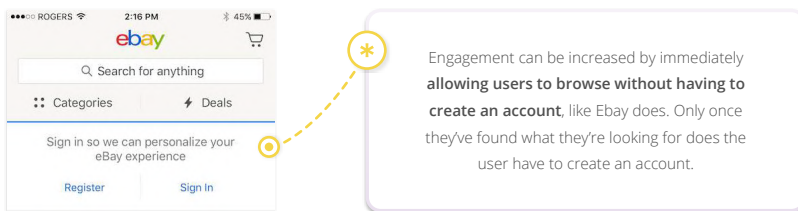
Engagement is the amount of time that your users spend inside your app. You want to create a connection with your user so that they will become loyal. The sooner you can get them hooked and make a habit out of your product, the better off you will be.

Onboarding

Onboarding is mentioned again because of its importance in developing loyal, engaged users. It's specifically important to engagement because you need to find the balance between engaging them quickly and gathering enough user information.

Try experimenting with:

- Providing the option to skip registration
- Immediate challenges or incentives
- Optional product tours

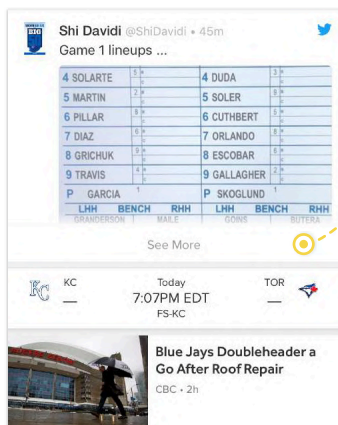


Personalization

Personalized messages increase conversions by 27% compared to generic messages. It keeps them more interested and spending more time engaging with content because it appeals to their interests and preferences.

Some ways to experiment with personalization in your app:

- Push notifications and communications sent via email
- Homepage or marketplace layout
- Autofill settings or obsolescence of form fields



By asking users to pick their favourite teams when they sign up, **Bleacher Report creates an engaging feed that is customized** to news, updates and standings that are interesting to the user. This will create higher engagement, since the user is more likely to find content they want to consume.

Segmentation

One way that experimentation can uniquely help you create hyper-personalized experiences is through segmentation. Experiments can be segmented to be sent to any custom groups of users you like to discover how different types of users react differently.

Want to know how users onboarded in last month reacted differently to a retention push than old users? Or how different deals appeal to shoppers under the age of 25? You can handpick what criteria qualifies a user to receive your experiment to find out!

Some segmentation ideas:

- Levels of engagement - sort by the last time they visited your app, or frequency of visit
- User demographics - age, gender, location, time zone
- Length of Membership - How recently did they join?

Conversion

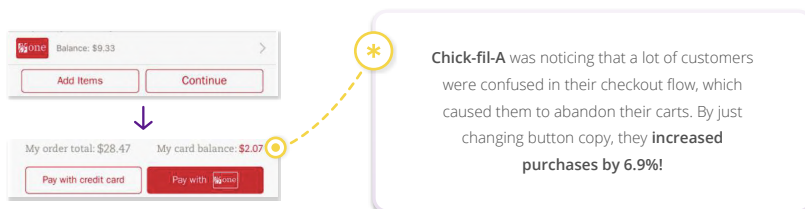
You want as many people to get to your app's aha! moment as possible. Whatever it is that completes their user journey (and defines your success!) can look completely different depending on what kind of company you are - making a purchase, clicking that CTA button, signing up, etc. Experiment with some different techniques that will get more people to cross your finish line.

Checkout Flows

Cart abandonment rates in apps can be as high as 82%. Find out what it is that has your users abandoning ship at the last minute by running tests on the checkout flow.

Try experimenting with:

- Number of steps or screens
- Methods of payments & autofill or autopay options
- CTA button copy



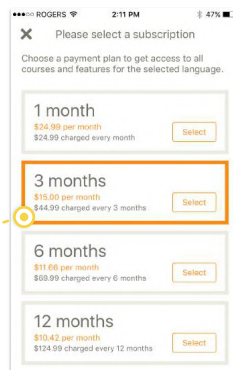
Pricing & Upgrade Pages

For some apps, a key moment is the conversion of a freemium user to a paying one. Experiment with the ask and the way that the information is strategically presented. Remember: it's all about them, so make sure you express the value they'll get from the upgrade!

Try experimenting with:

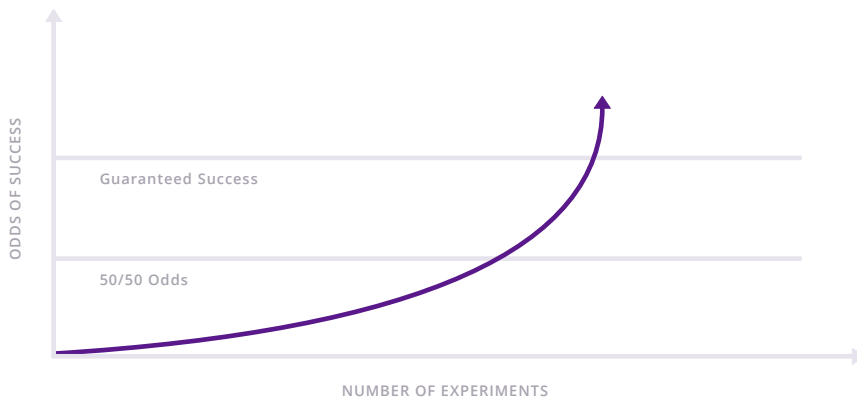
- Timing of the ask - Is it a requirement once a certain level is reached? Is it a pop-up?
- The layout of the pricing and plan options
- Messaging and value proposition

On **Babbel's** pricing page, they highlight the 3 month option to catch your eye. Additionally, they highlight in orange the cost/month to **incentivize the purchase of a longer plan.**



Rule Of Thumb

Simmons' 10,000 experiment rule simply states that "deliberate experimentation is more important than deliberate practice in a rapidly changing world." Avoid making large, massive bets on one or two big projects. Instead, build the habit of running smaller experiments at a faster cadence.

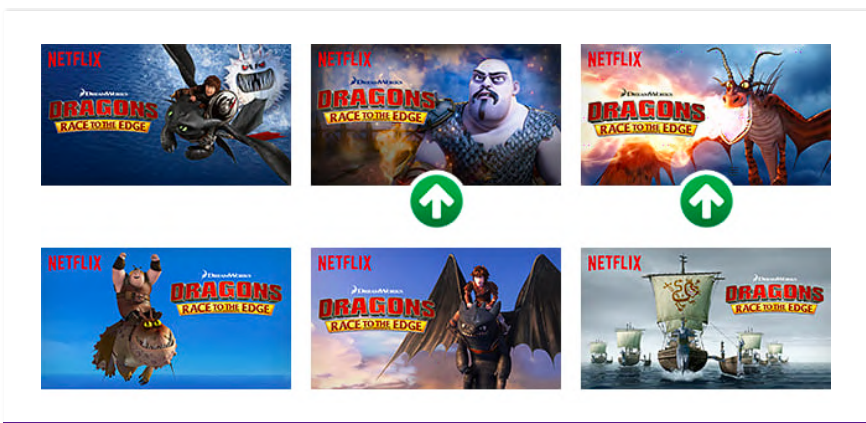


Netflix researchers estimate that if a typical user doesn't find something to watch in the app within 60-90 seconds, they run the risk of getting bored and moving onto something else.

By following an empirical approach, we ensure that product changes are not driven by the most opinionated and vocal Netflix employees, but instead by actual data, allowing our members themselves to guide us toward the experiences they love.

For example, the data team at Netflix found that users look at the artwork first before deciding whether to click for more details around it:

- First, they experimented with a simple A/B test to see if they could increase engagement by changing up the artwork by measuring click-through rates, play duration, etc.
- Next, they wanted to see if changing the artwork would contribute to increasing total streaming hours across the product. They tested to find the best artwork for each title over a period of days, then served that artwork to future watchers to see if that would result in a higher number of hours streamed.



How To Build An Experimentation Culture

While it's trendy to pay lip service to principles like “test don't guess” or “move fast and break things,” it can be hard to get your organization rapidly experimenting and optimizing —especially when it runs against the way things have been done for years.

Here are four steps you can take to create an experimentation culture:

1 Step 1 - Start Small

To map out these tiny steps, move backward from your major goal. Identify:

- Your high-level goal
- What needs to happen to achieve this goal
- An A/B test you can implement to meet your goal

2 Step 2 - Practice Good Experiment Hygiene

Having an established testing protocol keeps your team's test results accurate and meaningful, regardless of who's controlling the experiment.

- Test one thing at a time. With more than one variable, your test results aren't meaningful because you can't pinpoint which factor is affecting your tests.
- Know how to measure your results before the experiment begins. Your tests are useless if you're unable to collect your experiment results.
- Establish a time frame. Think about the requirements of your experiment, and set a time frame long enough to collect enough information to form insights.
- Set a regular cadence for testing so it becomes a consistent part of your team's patterns and thinking.

3 Step 3 - Communicate Results Back To Your Team

By sharing experiment results, your team's testing becomes sustainable. Constantly learning from each others' feedback, your colleagues have plenty of insights for building and improving new tests in the future.

Not all A/B tests have clear winners or drive noticeable spikes in revenue or adoption. Continuously testing small changes is the best way to optimize your site without negatively impacting important metrics along the way. Sometimes, testing is about learning what not to launch—so share learnings from the tests that cause “failures” as well as the winners.

4 Step 4 - Set A Regular Cadence

Regularly analyze test results and communicate with others - Schedule time to look at test outcomes and establish methods or channels for sharing insights with the key teams so they can implement changes accordingly.

Schedule quarterly meet-ups for ideation & reviewing data - Have a quarterly meeting to sync on testing priorities and review high-level experimentation insights with key stakeholders from across your organization. This will help you make an effective testing plan, maintain internal buy-in for A/B testing, and prioritize your experiment roadmap.



Introducing No-Code Experimentation To Increase Speed

Taking a product from an idea all the way to release requires input from team members throughout your organization. Being able to collaborate effectively across these teams is the key to making product development as seamless and valuable as possible.

Then why are 75% percent of cross-team collaboration efforts dysfunctional? Because navigating the various pitfalls and potential issues that arise when working cross-functionally isn't always easy. It requires a culture of open and honest communication, as well as a strong strategic approach to creating value for customers.


Fostering cross-team collaboration not only helps increase productivity but also frees up resources for your team and makes innovation easier throughout the development process. And a critical piece of collaborating well as a team is being able to experiment and test your hypotheses, which is where no-code experimentation can help.

When your team understands how they work together and has the tools and autonomy to refine these processes as they move forward, that makes achieving your overarching goals simple.

No-Code Experimentation Frees Up Resources For Your Team

No-code experimentation facilitates cross-team collaboration because it removes one of the biggest barriers to product development for companies that lack enterprise-grade budgets: finite engineering and development resources. When your product and marketing teams can easily run tests on their own, it ensures that developers stay focused on adding value through their day-to-day tasks.

Previously, only companies with the resources to build in-house experimentation tools



could quickly deploy tests and learn what successfully engaged users or led to better conversion rates.

Thanks to no-code testing tools like Taplytics' cross-channel A/B testing, every team within your organization can experiment collaboratively without the need for development resources. This allows product teams to collaborate with their marketing counterparts to deploy tests across websites, apps, and other connected devices. And it provides actionable data they can use to inform product decisions in the future.


Being able to test autonomously frees up product, sales, and marketing teams to focus on their individual goals as well. Instead of relying on engineering and product teams to implement and execute tests, they can increase experiment velocity and any learnings to analyze results to better understand how those experiments increased conversions or reduced user churn.

“The best approach will vary from one company to the next, but at its heart lies the challenge of transcending functional boundaries—a difficult achievement in itself,” said Paul Leinwand and Cesare Mainardi in their book, *Strategy That Works*, an analysis that included studying cross-team collaboration. “This means centralizing and systematizing activity throughout your company while still fostering participation and experimentation.”

No-code experimentation makes it possible for product, marketing, and engineering teams to systemize experimentation and build a culture of collaboration.

Cross-Team Collaboration Simplifies Continual Iteration

Combining no-code experimentation with a focus on cross-team collaboration helps you work through the product development process faster without sacrificing quality. Teams that have a clear sense of how they work together will grow and scale more effectively—refining processes and workflows to deliver the most value possible to your customers.



They're able to do all of this because each member of your organization, from the product managers defining the strategy to the marketers crafting messaging campaigns, knows how to work together. They understand the underlying goals and desired outcomes of every release, as well as the best way to communicate your product's value to potential customers.


Great cross-functional teams also understand that continual iteration and release are built on top of a base of solid data—data that informs decisions and guides experiments. And they know how to run these experiments autonomously. This not only frees up developer resources to focus on building new and exciting products for customers, but also helps each member of the team engage with their work on a deeper level.

But improving your teams' ability to iterate through the product development process is just part of the equation. Effective cross-team collaboration cuts down on total development time and helps you generate value from each release faster.

No-code experimentation gives your team the ability to deploy multiple experiments simultaneously, which avoids the time-consuming process of deploying experiments successively. When you combine this with the ability to test across multiple channels, it ensures that each stage of the experimentation process is as streamlined as possible.

The ability to experiment faster across each of these channels makes collecting data and proving hypotheses easier, which helps your team create a higher converting product while improving the overall customer experience with each test.

Let's say you have an idea for a new user onboarding flow, one that originates from within your marketing team. That team documents their idea and the potential impact they think it will have on the user experience and ships those details over to the product team.



The product team mocks up the new experience in wireframes and fleshes out potential designs and requisite features. Once the idea is integrated into the overarching product strategy, they define the requirements and send the project to engineering for active development. Each stage of the product development life cycle takes time and resources to complete, all of which are factored into the total time-to-value (TTV) for your project.

When you're able to cut down on the time it takes to create value for your business, that helps build a clearer picture of the impact your team has on the company's bottom line. Using that knowledge, you can create more effective revenue and resource projects for the future.

Cross-Team Collaboration Drives Innovation

One of the biggest benefits of cross-team collaboration is that teams from different disciplines, backgrounds, and skill sets can work through innovative new ideas together.

To facilitate this kind of bottom-up innovation, try to foster connections between every member of your team. To do that, you need to have a company culture that's built on trust and a shared understanding. Encouraging cross-team collaboration ensures that each member of the team feels like their opinions are not only valid but that they're heard as well. Even the kernel of an idea can become a valuable and engaging product, given that it's visible to the entire team.

No-code experimentation further helps drive innovation and new ideas by enabling teams to quickly apply their hypotheses to experiments. That means they can validate their assumptions directly—bringing more fleshed out and interesting ideas to the rest of the team. When collaborating teams can quickly see conclusive results to experiments, that path to innovation is more straightforward, as winning ideas can be validated and replicated with data rather than guesswork.



No-Code Experimentation Makes Collaboration Easy

No-code experimentation gives organizations of all sizes a way to improve cross-team collaboration. It offers smaller organizations an effective shortcut that makes it possible for cross-functional teams to learn, engage, and innovate more quickly without the need for commitments of developer resources.

When cross-functional teams deploy rapid testing via no-code experimentation, every team, not just those with the budget and resources of an enterprise-sized business, can optimize experiences for their users.

No-Code A/B Testing For Mobile Apps

When resources are tight, experimentation is right

Nothing is worse than being responsible for your mobile app's KPIs and growth, but unable to do anything about it because you're tight on resources, and don't know a thing about coding to boot. But what if we told you the experiments in this guide can all be run by non-technical team members to optimize key areas in your mobile app? That's right, no coding required. At all. And they have all been proven to move the needle on metrics like mobile engagement, retention, click-through rates, and conversion.

So, here's the thing.

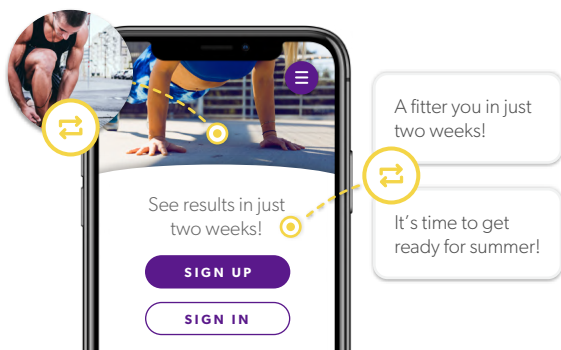
These experiments really can be run with no coding at all, just as soon as your engineering team installs the Taplytics SDK. Be sure to treat them to a nice lunch and give them the heads up that our SDK is lightweight and secure. The installation will take less than an hour (we promise!) to do. Et voilà! You now have access to a powerful visual editor that lets you run experiments with a tap and a click. Looking for some inspiration to get started? Let's go through some of our most successful experiment suggestions, which have been validated by our world-class clients.



Throughout this section, keep an eye out for tips from **Kate and Dexter**. They're our in-house testing experts who lead organizations through high-impact experimentation programs with Taplytics Enterprise Services.

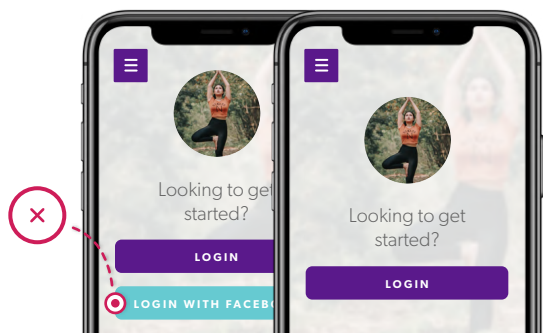
1 Experiment 1: Optimized Onboarding Screens

Your onboarding screens are the first elements users will interact within your app, so naturally, this will be one of the highest impact experiments that you can launch. Optimizing this first experience can affect the number of users who complete the flow and engage after that first session. Taplytics customers have seen registration and purchase increases of 150% and 9%, respectively, by running onboarding flow experiments proving how valuable an optimized onboarding screen is.



* SWITCH UP YOUR SPLASH SCREEN

When a user opens your app for the very first time, what images and copy do they see?

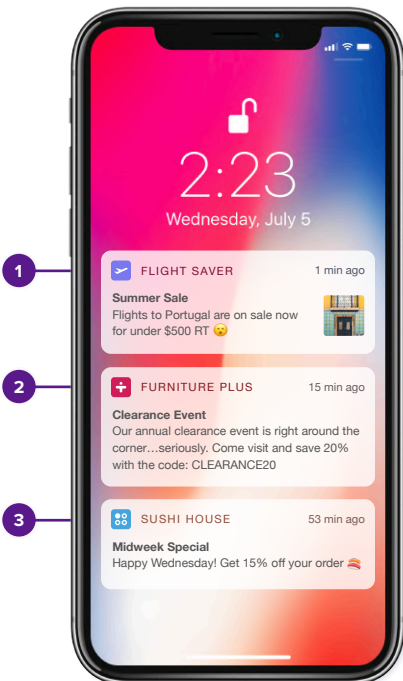


* HIDE ELEMENTS

Find out if providing additional on-screen elements is distracting or confusing for users.

2 Experiment 2: Push Notifications

Push notifications offer powerful ways to share compelling and actionable information with your users to re-engage them with your app. The problem is, mobile users are bombarded with dozens of pushes every day. By running a push experiment campaign, a Taplytics customer saw a 133% lift in engagement for those users who received the push. You can get creative in new ways and experiment with your pushes to see what makes you stand out from the noise.



1 EXPERIMENT WITH CAPTIVATING CONTENT

Try fresh copy, emojis, and even images in your push notifications.

2 TRY GEOLOCATION TRIGGERED PUSHES

Greet users with a relevant offer or message when they step into your store or within a specific proximity.

3 TEST THE TIMING OF DELIVERY

Do your users want to hear from you in the morning or the afternoon? It's easy to find out!



"Always align your timing with the content you're pushing - sending a lunch deal at 11:30 on a Friday is a perfect example."

3 Experiment 3: Conversion Elements

Little details can have a significant impact when it comes to the copy and styling of a call-to-action. What language persuades your customers to tap the buy button? Is it “Buy”, “Buy it Now”, or “Add to Cart”? What about the button color? Should it be red, green, or yellow? Even a 2% lift in purchases from a simple copy change is nothing to scoff at and can be done in an instant with our visual experimentation tool. By completing a simple CTA experiment, Taplytics customers have lifted click-through rates by up to 60%.

Get unlimited access with NewsNow Premium!

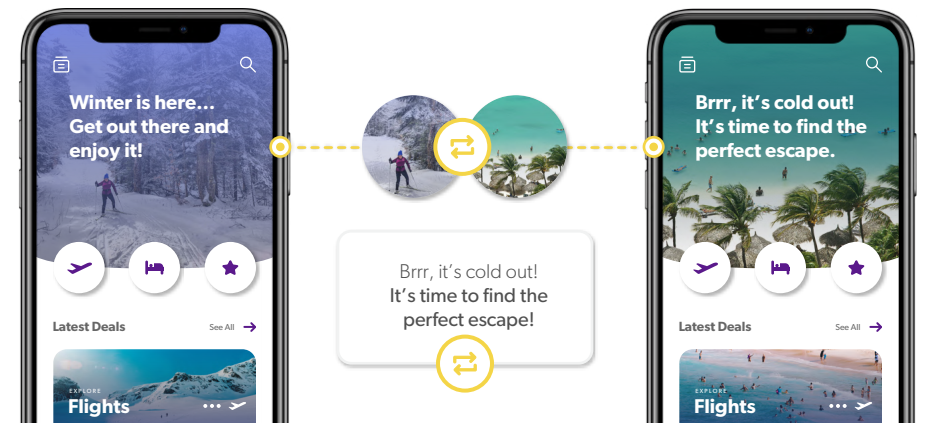
- * **WORDSMITH YOUR COPY**
Titles, paragraphs, and taglines are all fair game for quick experimentation.
- * **EXPERIMENT WITH BUTTONS**
Change the color, shape, size, and copy of your CTAs to dramatically improve conversion rates.

4 Experiment 4: Aesthetics & Visuals

Your app should be eye-catching and intuitive to navigate, and different visual representations of your app will appeal to your users more than others. A few simple visual changes can give your app a completely new look and feel that might resonate (and convert!) better. Makeovers don't have to be complex – remember, these are all no-code changes being made to the text, images, and buttons. Taplytics customers have seen conversion improvements of +30% from visual experiments. If you're nervous about including your entire user base in an experiment, that's okay! You can segment experiments so they're only set to users with a specific demographic, join date, device, and more so you can slowly roll-out the changes.

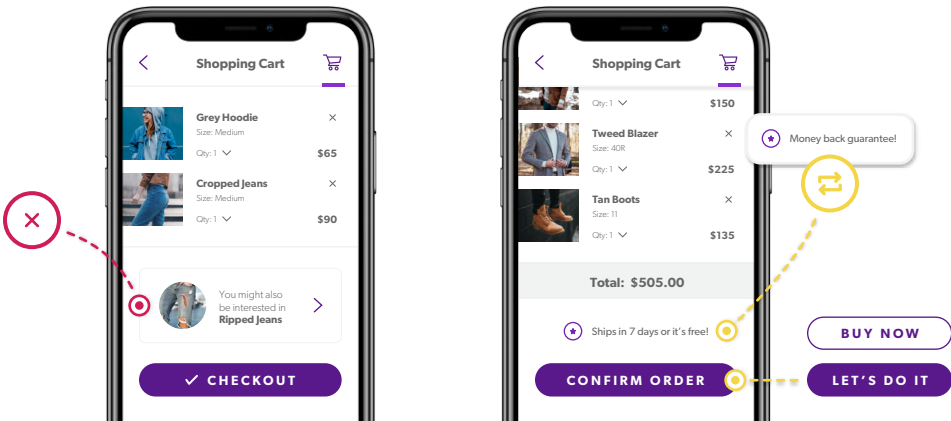


"If you're nervous about including your entire user base in an experiment, that's okay! You can segment experiments so they're only set to users with a specific demographic, join date, device, and more."



5 Experiment 5: Checkout Clarity

When it comes to mobile checkout design, you care the most about getting shoppers to complete their purchase. We're often tempted to make a last-ditch effort to increase their cart size or value, which can sometimes be harmful. Experiment with what, and how much, content you're presenting in your checkout flows to maximize purchases. By simplifying the checkout experience through experimentation, a Taplytics customer saw a 6% increase in purchases and eliminated all customer service inquiries at checkout.



* HIDE ELEMENTS

Sometimes less is more. Identify if extra text or offers enhance or distract from your users' checkout experience.

* TRY NEW TEXT

Which offer, guarantee, or promise will push the most shoppers over the finish line?

Putting Your Experiments Into Action

It's hard to believe that all of these high-impact experiments can be launched without a single line of fresh code. All of the examples you saw can be created in the Taplytics Visual Editor or Push Notification Dashboard and shipped in less than a day.



No-Code Web A/B Testing: Easily Convert More Customers

Experiments aren't just for people in white coats. If you're looking to boost your product landing page conversions, A/B testing your landing page can have a massive impact. HubSpot increased its blog subscribers by a whopping 128% simply by testing a new checkbox on their landing page forms. That's a huge payoff from such a small change.

The concept of A/B testing is simple. It involves displaying two versions of the same web page to visitors to determine which converts better. More often than not, marketers don't take the time to experiment and run A/B tests on their landing page. According to HubSpot, "Only 17% of marketers use landing page A/B tests to improve conversion rates." That's a surprisingly low number when you consider that landing pages have the highest conversion rates of any type of sign-up form (23%).

Because web A/B testing can have such a tremendous impact on lead generation and conversion rates, here's how to optimize your product landing page, along with some highly effective web A/B tests you can use right away.

What Makes A Great Product Landing Page (And Why They Matter)

Research by Omnisend reveals that landing pages are the least popular type of sign-up form with marketers, yet they have the **highest conversion rate** (23%). Meanwhile, pop-ups, the most popular type of enabled sign-up forms, have really low conversion rates (only 3%).

This leads to two important conclusions:

- 1** Your product landing page represents one of the most powerful tools in your customer conversion tool kit and is worthy of your time and investment.
- 2** Every single element on your product landing page should be carefully considered and, ideally, A/B tested for its effectiveness.

While every landing page is different, five basic elements go into every successful landing page.

An Eye-Catching Graphic Or Video

Your landing page image is a high-quality image, video, or animation that clearly and immediately conveys the core message of your product or service. While landing page images are often product photos, they don't need to be a literal image. For example, a picture of a cheetah sprinting could convey the speediness of a file transfer service far better than a ho-hum picture of a server could. This is something you could A/B test to see which image or video performs best on your website.

An Attention-Grabbing Headline

Your headline is a short, attention-grabbing statement that aligns with the message of your campaign. They're to-the-point messages designed to provoke thought or empathy or matter-of-fact statements, like this one: "You get fresh, hot pizza delivered to your door in 30 minutes or less—or it's free." To ensure your users see value in your offering, it's important to master the basic principles of headline-writing.

Clear Value Propositions For Your Product Or Service

Your value proposition is the problem your product or service is designed to solve. It's what makes you better or different than the competition and explains the exact benefits consumers can expect to receive. Landing page value propositions should be succinct

and easy to scan and absorb. When creating a value proposition, always focus on the benefits your product or service offers, not the features.

Social Proof From Your Customers

Website visitors frequently look for social proof before making purchasing decisions. Statistics from HubSpot show that a whopping 88% of consumers trust user reviews as much as personal recommendations, which means you can leverage social proof and testimonials to great effect. Forms of social proof include online reviews, customer testimonials, industry certifications, and logos, ratings (four out of five is best), and celebrity or expert endorsements.

A Compelling Call To Action (CTA)

Last but not least, your CTA is the critical touchpoint of your landing page, the place that encourages your visitors to click and take the next step. Usually, a landing page CTA is an easy-to-find, colorful button with copy that ties into your original value proposition, such as, “Build my pizza” or “Start my free trial.” Avoid generic words like “Submit” or “Go.”

How To A/B Test Your Product Landing Page

To get started with web A/B testing, just follow the five steps below. Keep in mind that you’ll need an experimentation platform like Taplytics, which enables you to set up and run your A/B tests as well as analyze the results.

1 Analyze Your Current Performance

How well is the landing page performing, and where would you like to see improvements? For example, you may want to increase your CTA button conversions by 20% or get more people to download your ebook.

2 Create A List Of Elements You Want To Test

This might include new writing copy for a button, changing the color of a button, adding a new checkout feature, etc. To identify which element is driving the results of your A/B test, test only one element at a time.

3 Set Up Your Version A & Version B

Version A is your normal landing page with no changes, and Version B is the test page with the new element you want to test. Version B should contain only a single changed element. To test additional elements, create a new test page for each one.

4 Run Your Web A/B Tests

To run your web A/B test, you'll need an experimentation platform, like Taplytics. Taplytics enables you to set up your landing page experiments over web, mobile, and cross-channels and track the results in real-time.

5 Analyze The Results

Again, you'll need software to do this. Taplytics software enables you to track your A/B test results as they happen and provides a suite of comprehensive tools so you can analyze the results and schedule rollouts of the best variants.

What Elements To A/B Test

CTA (Call to Action)

- **Change your CTA button color.** Test different colors on your CTA button. Red, orange, and green are favored by marketers, but which color is best is still open to debate.
- **Change your CTA button text.** Experiment with different CTA messages. Try to avoid generic messages, such as "Submit" or "Go."

Copy (Headlines, Subheads, Descriptions, Etc.)

- **Try different headline text.** Test out different headlines and A/B test them to see which ones are most effective.
- **Add a subhead.** Try adding a compelling subhead under your main headline to give the reader more information.
- **Update your product descriptions.** Update your product descriptions to include more compelling value propositions.

Offers & Incentives

- **Test different shopping incentives.** These might include free shipping, a money-back guarantee, a more generous return policy, etc.
- **Experiment with different offers.** Some options include free business templates, free samples, a complimentary ebook, a free white paper, etc.
- **Add a limited-time offer.** Create a sense of urgency by offering a benefit for a limited time only.

Checkout Options

- **Add new payment options.** Make it easier for customers to check out on your website by offering a range of checkout options — for example, PayPal, Google Pay, Amazon Pay, cryptocurrency, etc.
- **Offer a guest checkout option.** Make it easier for new customers to check out with a guest checkout option. Research shows customers are 1.2 times more likely to choose guest checkout than logging in.
- **Add a security badge.** A recognizable security badge can help reassure customers and push them over the line into a purchase.

Imagery

- **Change up your images.** Experiment with different image types (photos, graphics, diagrams), sizes, and locations on the landing page.
- **Compare an image vs. a video.** Test the difference in effectiveness between a static image vs. a video or animation.

Iconography

- **Change your icons.** Experiment with different symbols. Icons should be instantly recognizable (not too cryptic) and help customers to navigate better.

With Taplytics Web A/B testing, you can easily set up tests for all of the above variants on your product landing page, track the results in real-time, and analyze the results — all without writing a single line of code. We make it simple to measure the impact of your experiments with flexible goal-tracking and integrations, while our AI assistant can suggest areas for improvement.

FAQ: Client vs Server-Side Testing

Experimentation cultures are built upon a willingness to test, iterate, and optimize product experiences to the needs of the customer. The question for product, marketing, and engineering managers becomes - how do you go about doing that?

There are pros and cons to both options, and your team should decide:

- Why you need to run experiments
- What types of tests you want to run
- How you should measure the effectiveness of those tests

How do you run experiments with a small engineering team?

There's a running myth that companies with large engineering teams have enough resources and bandwidth to do more complex forms of testing. Those myths strongly hint that server-side testing is the optimal solution in those instances.

However, while there is some truth to that myth, the reality is that it all depends on how much bandwidth there is within your engineering department. It helps each product or marketing team to determine the complexity of the experiment in question, and then align with engineering on whether this particular test is something that will require a lot of their time.

If the answer is 'yes, it will require a lot of their time,' ensure that there is enough bandwidth among the engineers who are very familiar with the ins and outs of the server to implement a proper server-side experiment. On the other hand, if the answer is 'no, those experiments don't require a lot of technical support,' it would benefit the product/marketing teams to implement more forms of client-side testing. This allows

the lighter, less technical experiments to run independent of the engineering department and allows engineers to focus solely on the elements of the experiment that require their support.

What's the ideal length of time to run experiments?

This is another 'it depends' type of question.

Some of our clients have been able to build out all of the requirements for a server-side test in as little as 3-5 days of development work. In other instances, where the experiments are more complex, the setup time has required a full sprint to implement.

In terms of running the experiment itself, the minimum amount of time recommended for either a server-side or client-side experiment is 2 weeks. This gives you enough leeway to monitor user behavior on the page or in the app over both weekday and weekend periods, allowing you to analyze peaks and lulls in traffic.

The key component to consider is the size of your user base. To draw real conclusions from the experiment, you need enough traffic volume to understand what works and what doesn't. If there's a lot of activity on your site, you can draw those conclusions closer to that 2 week experimentation period. On the other hand, if activity is low, you may require more time to draw concrete conclusions.

How do you run cross-channel experiments?

The ability to manage SDK devices is a key variable in answering this question. You want to make sure that your team is sufficiently capable of implementing and monitoring experiments across all of these devices to get a real cross-channel understanding of how users respond to your experiments.

Suppose you have dedicated resources to individually support web, mobile, and OTT experiments. If that describes your situation, you can probably manage those various

tests using a client-side solution.

However, in this particular use case, server-side testing could be the simpler solution. A team that can implement cross-channel experiments directly from the server can simplify the development process, the resources required to build the experiment, and capture all of the user behavior data in one centralized location. If your team has the bandwidth to build a server-side experiment, this would be the ideal approach to a cross-channel test.

Why is the “flicker effect” such a concern?

Historically, client-side experiments can cause what’s known as the “flicker effect.” This is a result of a live webpage or in-app experience rendering on a device, and then suddenly becoming overridden by the elements of the experiment.

When a device is loading an experience, a network request is sent to the server that hosts the experience to render the on-page layout onto the device. But a client-side experiment sends a second network request to pull in the elements of the experiment itself. This second server request is what’s known as the “flicker effect.”

Users will notice the change if messaging or visuals suddenly update on the second server request. There may also be an effect on page or app loading times due to the multiple requests sent to the server.

Modern tools have virtually rendered this flicker to be non-existent; however, the conversation is still relevant as it pertains to site conversion and SEO. A 1-second speed improvement to your site or native app can result in over 25% more conversion which puts an even greater emphasis on the need for lightweight, high performing tools - either client or server-side.

What kind of SEO impact is there from either form of experimentation?

SEO implications are a common question raised by our clients, particularly when it involves web or mobile site tests. There are notable differences between client-side and server-side testing and the SEO effect.

For client-side experiments, the good news is that changes to the javascript on the page typically are not indexed by Google during the experimentation phase. That means the experiment itself is unlikely to appear in or have an influence on SERP rankings. The content within the experiment will only become fully indexed if it's rolled out in full across the site experience.

Site speed is a critical component of Google's decision to rank sites appropriately. As a result, client-side testing can impact site loading times which, by extension, can impact your SEO.

In contrast, server-side testing has no impact on site loading times. However, since the experiments are implemented at the server level, they are far more likely to become indexed by Google. This means that your experiment could render in SERPs and indirectly influence SEO.



The Modern Product Manager's Guide To Continuous Deployment With Feature Flags

There's a lot of risks involved in releasing a new product. Not only do you need to release on time, but you also have to ensure that the experience is as seamless as possible for your customers and your team. Lagging behind schedule due to poor release management or causing unnecessary downtime due to bandwidth or system constraints can have a direct and lasting impact on your relationships.

That's why it's so important to maintain as much control over the release process as possible—which is where feature flags come in. This tool allows you to roll out or rollback features easily, so you're able to minimize any negative impact on your customers and team.

They're also at the core of continuous deployment. Without feature flagging, it's challenging to achieve the right balance between speed and quality inherent to the continuous release process.

When you understand what feature flags are, how to implement them, and why they're so powerful, it's easy to create a better release experience.

What Are Feature Flags?

Feature flags (also known as feature toggles) are a software development technique that allows you to turn specific functionality on and off without writing additional code. They provide you with the ability to make dynamic changes to your code without the need to deploy a new version.

These dynamic changes give you more control over how and when you release your new product or feature. Using feature flags means it's easy to:

- test features in production,
- collect actionable user feedback,
- control the individual user's experiences, and
- get more visibility into the impact of your release.

Used correctly, feature flags are one of the most high-leverage tools you have at your disposal for any upcoming release. Releasing small batches of code as soon as they're ready is much less risky than big feature releases.


The key is finding a way to build a culture of constant deployment around these constraints. Building, shipping, and iterating features quickly without compromising on quality enables better product development. You constantly learn what people want through your releases and can easily make improvements to keep users happy.

Feature flags provide the agility needed for constant, high-quality deployment. They allow you to launch features in real-time to a subset of users so you can quickly and safely resolve potential issues before fully releasing them to all users.

How Do Feature Flags Work?

Feature flags enable, disable, or hide specific features for testing or release. They give you a way to manage your application's behavior for testing or release with select user groups. Whether you're making changes to your codebase, your website, or the functionality of an individual process, feature flags can help.

Let's say you're planning a redesign of your product and want to roll it out to customers slowly so as not to impact their overall experience with your service. Feature flags give you the ability to control which users are first to see the redesign, when they'll be able to see it, and how long it will take for all users to have access to the feature.



In the event of issues, such as your application crashing due to increased user interaction, you can roll the feature back to its previous state immediately. Think of it like flipping a switch.

There are two main functions of feature flagging for development and product teams. While marketing teams can make use of this functionality to A/B test site design and messaging, feature flags provide more direct value to developers and product managers.


Feature Flags Enable Continuous Deployment

Feature flagging is the key to continuous development. They are the software development technique that gives engineers the ability to create branching production trees—the underlying functionality that helps you develop on an individual feature, or aspect of a feature, without impacting the code that’s live for customers. Feature flags can also be used to run tests with specific segments of your user base while in production.

This image outlines the branching process, where developers work on an upcoming feature alongside the master branch or production branch. Whenever you create a new feature, this practice ensures you’re not making changes to the customer-facing product without full confidence in its security and usability. You break off from the main trunk of your production pipeline to create the feature and merge it back once it’s complete.

Starting at the “common base” in the example above, developers can use a feature flag to split off an additional branch. This allows them to actively develop the new feature without making changes to the codebase their users interact with.

Once the developers are done working on the feature, they’ll merge it back into the master branch, essentially turning off the feature flag they created in the first place. This brings the code they created on the feature branch back into production.



The more developers you have working on a specific feature, the more complex feature flagging becomes.

It's important to understand that while feature flagging helps create a more seamless experience for developers, it does add complexity to your system. Each new branch exists independently from the other until it's merged back into the master, so always make sure you're branching as efficiently as possible.

Inform Product Development

When product teams use feature flags for a product release, it helps you make small and impactful changes quickly. This decreases the overall time it takes to release a new product or feature and helps you provide more value to users on an ongoing basis. Combine this with the fact that you can easily test new features while in production, and it's easy to see how powerful feature flags can be.

Consider the way you normally release a new product or feature. Your planning and development process makes it easy to execute a release management plan, but any change to the overall experience of using your product can have a lasting impact on your relationship with customers. Once you're done planning and active development, the risk becomes all about how your upcoming change impacts the user experience.

Feature flags help you mitigate the risk of a botched release through a canary release, where you roll out a new feature to a small subsection of users first—before making it available to everyone. This phased release schedule minimizes the impact on customers by giving your team more control of the experience at every step of the process.

Feature flags provide a level of feedback that demos and other staged tests can't provide. Users are interacting with the new feature just as if it had been rolled out fully, so it provides a more accurate picture than a simulated test. You get to see your feature in a live environment.

Beyond the live environment aspect, feature flags also have the technical capability to provide more powerful insights than most testing. Greater than their basic on and off function, feature flags can have multivariate flags that allow users to be targeted at a very narrow level. They let you customize the number and type of variations returned to users so you can understand their preferences on a greater, more precise level.

Say, for example, your product is an e-commerce app. A multivariate flag could let you try different discount offers on users depending on their familiarity with your business. If the user is new, for example, you could set the discount to display as “30% off.”

Instead of just having an on-off toggle, multivariate feature flags allow you to employ conditional logic and exert granular control over deployment. You learn which variants will perform best with different user groups so you can release features fully with confidence.

Josh Wills, Software Engineer, Search, Learning, and Intelligence at Slack, agrees about the power of multivariate flags.



Josh Wills — Software Engineer @ Slack

“To me, the feature flag space is the parameter space that I get to explore to optimize whatever metrics that I want to optimize. If you are constraining yourself to this very limited boolean on/off space without strings, floats, etc. you’re putting artificial limits on how fast you can explore the space and about how all of the knobs at your disposal work.”

Providing powerful insights, feature flagging provides the necessary insights to deploy constantly. You understand which features will be the most successful with users, so you feel confident in gradually releasing more and more on a greater scale.

Reduce Risk

To build a sustainable product, you have to keep improving it with new features and functionality. But what happens when your “improvements” turn out to be mistakes that hurt your UX? If the feature has been fully released, every user is negatively impacted by the update.

Feature flagging eliminates that risk. It allows you to provide value to users with confidence because it makes changes easily reversible and controlled in distribution. You can do roll-outs of features to small test groups and easily disable the feature by turning the flag off instantly.


The system works through a check and balance mechanism. While the feature is only being released to a small subset, the feature is checked for bugs. If any issues are detected, the developers can use a kill switch to turn off the flag and stop the code from reaching users.

This control and reversibility of feature flagging enable top development teams to constantly deploy changes without worry. They can safely try out ideas in a live environment without the risk of affecting all users if an issue occurs.

Ship Faster

Before flagging, companies would wait years before releasing features. They needed a lot of time to prepare because if there was a mistake with a fully released feature, it would take quite a bit of time to rewrite the code. Netscape, for example, took four years to rewrite code for Mozilla.

Mobile developers today still have to wait before fully releasing features for app store approvals. These reviews generally take over a week so teams can’t immediately release changes to their app.



With feature flagging, companies don't have to spend this much time implementing changes. Flags can easily be turned on or off, so the company doesn't need to spend lots of time rewriting code if there's an issue. They can just disable the feature flag instantly.

With this on and off function, feature flagging speeds up deployment. Rather than spending time rewriting code after a faulty release or waiting for an app store review, developers only need to enable or disable a flag to turn a feature on or off.

The Mobile Flywheel Of Feature Flagging

Traditional, full feature launches on mobile face a major roadblock: getting approval from app stores.

Instead of being able to instantly implement changes, deployment is slowed down by the preparation and waiting involved in this approval process.

- The Apple review process takes an average of 8 days, a relatively long waiting period. Developers want to get new features out to users as soon as possible, so they spend a lot of time doing quality assurance and testing before the app review in hopes that their changes will get accepted on the first try.
- If there are any mistakes in your new feature, you have to send an update to be reviewed again for the app store to patch your product. This means waiting even more time before users can enjoy your new app feature.
- Users choose when and if to upgrade your app. Even when the app store approves your new features, you still have to wait for users to upgrade and enjoy the changes.

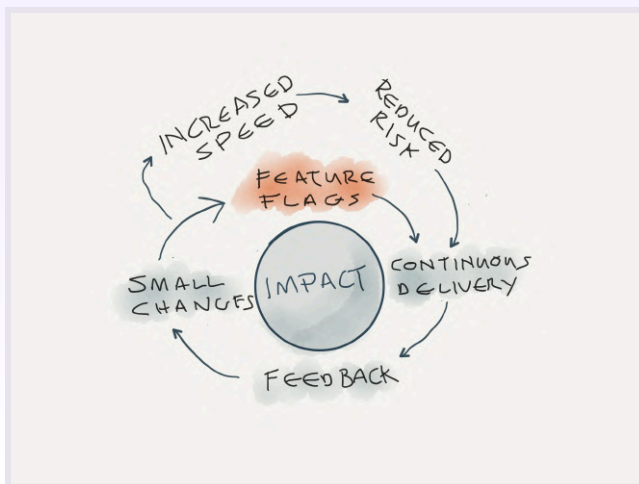
This slow deployment pace created by the approval process means you're not constantly getting user feedback on new changes to drive more iteration ideas—another factor that slows down deployment.

To speed up deployment, you can release features without needing app store approvals through feature flagging. This mechanism allows you to enable or disable features instantly and select when and which subsets of users see the features. Feature flagging drives continuous deployment because changes can be made:

- **Safely:** You can limit feature releases to small subsets of users. As you resolve issues and make improvements, you can gradually release them to more users.
- **Quickly:** You don't need to spend loads of time rewriting code if there's a mistake with your new feature—you just immediately turn the flag off.
- **Informed:** Feature flagging allows you to constantly get valuable user feedback that you can implement in further iterations to improve your app.

Together, these three aspects create an environment that drives continuous deployment.

Feature flags reinforce the benefits from small changes and continuous delivery, providing a flywheel to get you round the loop faster.




Features are released quickly and safely to user subsets. In a loop, this continuous deployment through flagging offers powerful user insights that drive even more releases.

How To Run A Phased Release With Feature Flags

To get started, have developers create the feature flags in your codebase first. Once they're created, your product teams can use them to target specific segments of your user base and roll out features with absolute control.

- 1** Determine the segment of your user base for the initial release. Whenever you release a new product or feature, there's a part of your user base you know will benefit from it the most. Determine this group by analyzing relevant user insights and using that information to identify users who would benefit from feature flags.
- 2** Build feature flags targeting those users. Once you understand what your users need, the problems they face, and how they interact with your tool, it's time to build a new feature that offers a solution. This is the active development process where you use feature flags to create a branch in your code.
- 3** Create a schedule for how the release will progress while you're in active development. This schedule should define the time it takes to finish development and test the new feature as well as your intended release date.
- 4** Define a rollout/rollback contingency plan. There's always the potential to run into issues during a release, so you need to have a strategy for how you'll react in that situation. Ensure a seamless experience for both your customers and your team with a contingency plan.
- 5** Perform the release. Roll out your new feature using feature flags from Taplytics. Monitor the progress of your release in case any issues arise and you need to follow your contingency plan.



Track the impact on your customers and team. Continuous deployments are built on the idea that more data leads to better overall product decisions. So once you've completed your phased release, do a post-mortem (we use easyretro.io) to understand any areas of opportunity to improve feature flags in the future.

Release With Confidence Using Feature Flags

Any new release comes with inherent risks, so it's important to find strategies that make you feel more confident throughout the process. Feature flags are one of the most powerful ways to manage your release more effectively. When every touchpoint during the release process can have a lasting impact on your customers, you need to have a process in place that minimizes risk.

With Taplytics, you can target specific segments of your user base, roll out and roll back features with the click of a button, and follow a timed-release schedule. This makes it easy to release with confidence, ensuring an effortless and efficient release for both your customers and your team.



Why Phased Rollouts Are The New Standard

Building a new product is a complex process with many potential points of failure. All members of your team need to work together seamlessly to move the project forward and ensure each task is completed in succession. Any delays can significantly impact your ability to launch the type of product your customers truly need.

A phased rollout helps you reduce the risks of these failures and makes it easier for product teams to plan and execute larger projects effectively. It forces product teams to examine their progress on a more consistent basis and make changes in real-time. By iterating continuously through the development and release process, everyone has the opportunity to increase their product's value and refine your release plan before it's in the hands of actual users.

Combine this with the increased control inherent in phased rollouts, and you're able to build a smoother product experience for your team and your customers. Each stage of the process will have clear goals and objectives and key results (OKRs) to measure your efficiency across the board. Without this clarity, you'll never be able to create an amazing product experience for your customers.

What Is A Phased Rollout?

Phased rollouts are the process of building and refining your product over multiple iterations. The core methodology helps you gain more control over various aspects of your development pipeline, from inception to launch day.

Building a phased rollout process for your team cuts down on the potential for scope creep by allowing everyone to reevaluate their workflow at every stage of the process. Looking at each phase together also helps you take a long-term view of the product experience, which you can use to define more effective deliverables at each stage.

5 Ways Phased Rollouts Helps You Launch Better

The phased development methodology helps you manage your product releases by breaking up each launch into smaller parts that are easier to execute. These more manageable components make it simpler to plan larger projects effectively and help your team see how their work impacts overarching goals.

When everyone understands the impact of a project on the macro level, it's easier to build a product that's truly and immediately valuable to your customers.


1 Accurate Planning

When you think about your product launches in phases, it lets you create a more realistic plan for your team. Each part of the product development process is represented as a phase, from ideation to active development, all the way through to release. Structuring launches in this way helps you leverage the underlying drivers that move your project forward.

In phased rollouts, each stage is functionally independent of the others. This makes it easier to budget required resources for each phase, whether you're referring to the monetary investment or bandwidth of your team.

To understand why, consider how many steps go into launching a product live to your customers:

- Customer research
- Project planning
- Goal setting
- Active development
- Quality assurance and user testing
- Code review
- Release planning
- Launch



Prospecting the number of work hours and the monetary investment required for all of these steps at once is a considerable task. But breaking up each process into an individual phase makes setting expectations significantly less complicated. You'll know that the research and planning phases will require more investment from your product and user experience teams, where active development and QA testing will need more developers and engineers.

Once you've adopted a phased rollout process, gauging how long each task will take is much clearer. With this clarity, you're empowered to communicate accurate timetables and requirements to your team from the start. This translates into a more easily executable product launch across the entire process.

2 Simple Prioritization

Iterating through each phase of a roll-out systematically helps you see how individual tasks relate to others in your product pipeline. This understanding allows you to prioritize which tasks need to be completed first based on their potential impact and helps you communicate those dependencies to your team.

Visualizing dependencies this way helps you manage the overall scope of your product launch better. Each task is represented by a block that spans the calendar and which team or individual is responsible. We can see here that the design team first needs to complete the homepage before the build team can get started on setting up their servers. This makes it easy to see how each phase affects the other stages of your project, making your determination of how to allocate resources much more concrete.

Whenever you complete an individual phase, it's simple to reassess your progress quickly and make changes when required. It's also a great way to communicate these priorities to your team.

3 Clearer Team Responsibilities

Product launches involve a lot of different moving parts, which can be difficult for teams to conceptualize when they're busy executing their individual tasks. Creating well-defined phases for any upcoming release makes it easier for your teams to see how their work impacts the project as a whole.

This increased visibility gives your team the context they need to hold themselves and their colleagues accountable. When you're working through larger projects with dependent tasks, this kind of individual accountability is key. Your team needs to trust that everyone else is working at the same level to move the project forward.

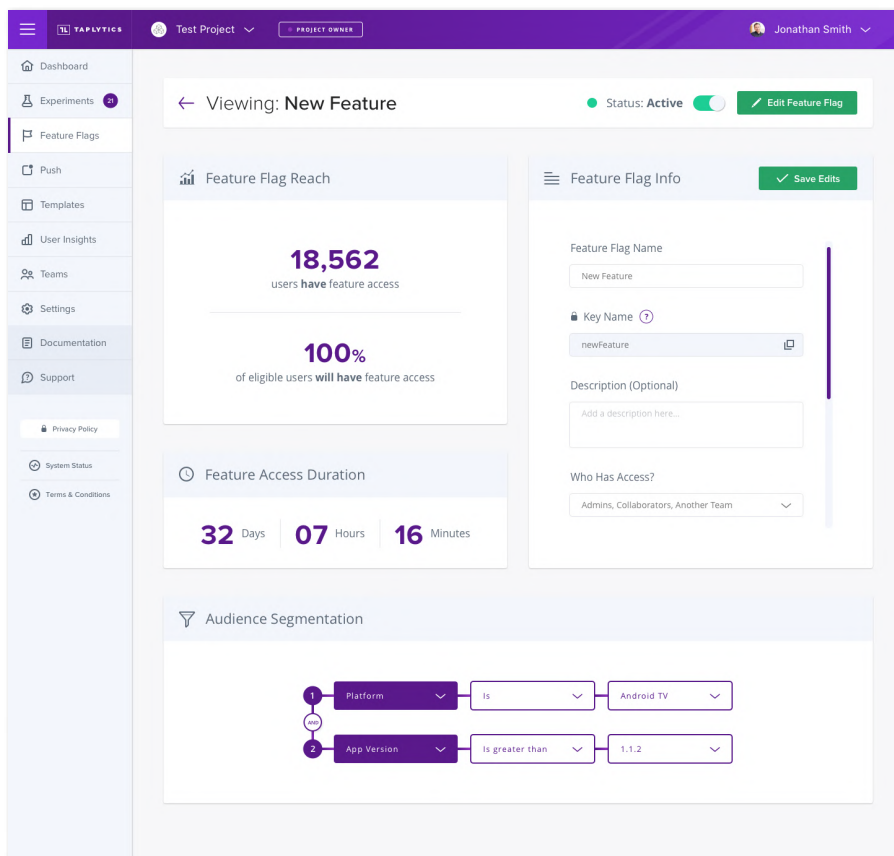
Think back to our Gantt chart example. If your developers lag on building the website template, that pushes backtesting and the launch date. Doing phased rollouts minimize the potential for these issues by clearly communicating how each part of the project works together.

An additional benefit of this clarification is that more visibility makes it easier to showcase wins throughout your team. Each team member will understand the work required to complete each phase of the development process and will see when people complete those early. This boosts engagement and creates a collective team ethos built on trust and shared goals.

4 Transparent Progress Tracking

With phased rollouts, you aren't expected to make perfect product decisions. When you create separate phases for all the work required in your product launch, it helps you assign more manageable KPIs and objects. When a phase is complete, you look at the overall progress of your project and make any necessary adjustments before moving on to the next stage.

Being able to make updates on the fly helps you refine your product before it gets to customers and ensure you're providing them with value.



Phases span a small unit of time, so any adjustments you need to make will happen faster. You're able to roll back updates and fix issues directly, without causing delays or putting undue stress on your team.

As you become more comfortable with the phased rollout methodology, you'll also be able to approximate how much time it takes to execute different types of work. This understanding helps you define more achievable OKRs and provide more accurate

timelines for your team. Each phase is an opportunity to assess your current process and make the necessary changes.

5 Reduced Scope Creep

According to a Gartner survey, 45% of product launches are delayed at least one month. The underlying reason for these delays often ties back to poor project planning and an unclear division of labor. Following the phased rollout, the method helps you better understand the scope of a product launch and minimize the risk of these oversights.

By building out small work units, you're less likely to overtax your team with too many requirements. Each phase is self-contained, so you've decreased the chances that one task will negatively impact another. And you gain more headspace to check whether deliverables are achievable based on your team's current bandwidth and time constraints.

This knowledge also helps offset the potential for poor time management, as you can adjust each phase to account for any potential issues as they occur.

Phased Rollouts Helps You Launch Better Products

The phased rollout process helps your product team execute projects with more clarity and direction. It empowers everyone to track their progress, make quick decisions, and launch products that are tied to users' needs. All this translates to a better overall experience and more visibility of why you've made certain decisions for your product.

When you're building out the product development process, visibility is the key to achieving the kind of shared understanding of customer and team needs that drives every project toward your overarching business goals.