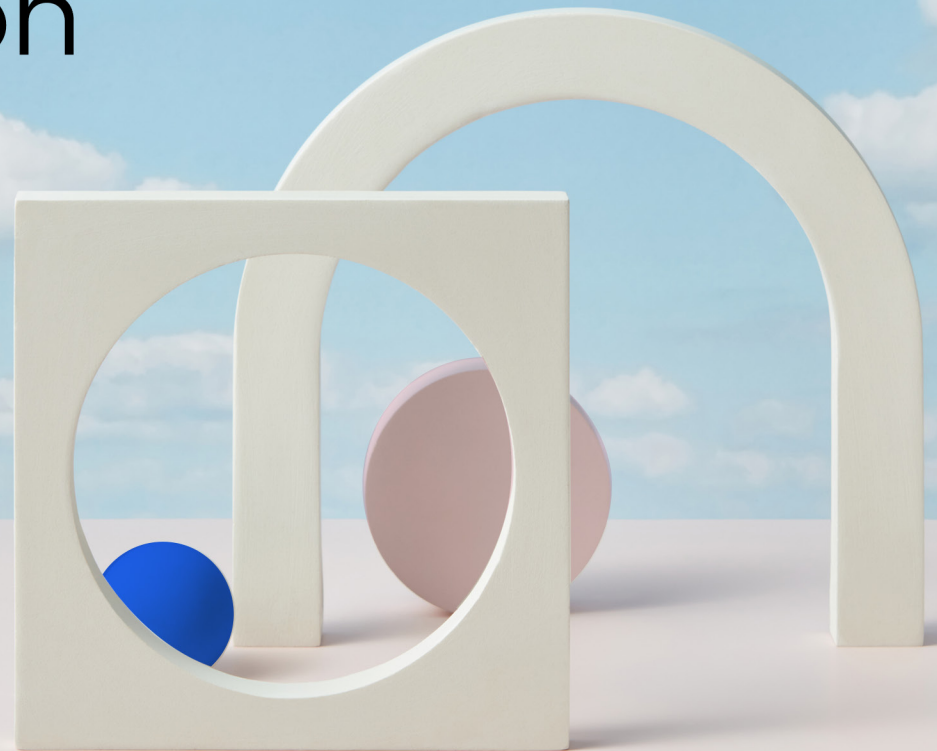


BUILD WITH CONFIDENCE:

Your Guide to Scaling Product-Led Experimentation



Co-Authored by

 Amplitude x  Reforge

In This Playbook

- 03** Introduction
- 04** Building a Culture of Experimentation
- 08** The Importance of Experimentation and How to Get Started
- 12** Experiment Ideation and Roadmap Development
- 17** Product-Led Experimentation for B2B Organizations
- 24** Feature Management and Experimentation: Two Sides of the Same Coin
- 29** Critical Capabilities for Product and Engineering Teams to Scale Experimentation Programs
- 46** Build vs. Buy: A Guide to Selecting Your Experimentation Platform
- 51** How Amplitude Experiment Guides You to Scale Experimentation
- 61** How Experimentation Works in Amplitude
- 71** Getting Started with Experimentation
- 72** Contributors





INTRODUCTION

In today's competitive landscape, product-led experimentation is more than a good idea—it's a necessity for product and engineering teams to quantify their impact. But they need to cultivate a culture of experimentation and infuse it into every phase of the product development process to build with confidence.

Built by experts and co-authored by [Reforge](#), this guide is designed to help product and engineering teams establish product-led experimentation programs successfully. Throughout this guide, our experts will walk you through how to build an experimentation culture, understand the practice of experimentation, prioritize your experimentation roadmap, and more.

Whether your goal is to make smarter decisions, maximize business impact, or drive rapid innovation, product-led experimentation can help. This guide will help you scale product-led experimentation to unleash the power of your products.



Building a Culture of Experimentation



By Saleem Malkana,
Reforge

Launch day for a new, major feature is always exciting. Bringing a squad together, creating something new, and getting it into the first customers' hands: it's product management at its best.

It's also only half the battle.

How do we know if we did the job? If we're successful? Many companies are happy when new deployments ship: presentation decks are updated, tasks are marked as completed, and teams are acknowledged in an all-hands meeting. But the process is far from complete when a new feature or product ships. In fact, shipping could be seen as a midpoint of your project. The best product teams know that what truly matters is not just shipping those outputs, but rather what outcomes they drive.

To identify and understand those outcomes, we need to be able to measure the effect of the changes we make to our product. To do this, teams must develop hypotheses, run experiments, evaluate results, and ultimately use those results to inform future strategy and decision-making.

Despite the quest for outcomes, the goal is not to ship more to maximize outputs. Instead, as product managers, we aim to maximize outcomes and minimize the outputs it takes to get there.

This is all a part of a culture of experimentation.

In this chapter, we'll explore what "culture of experimentation" means, how to cultivate it within your organization, and what success looks like as you build it.

The best product teams know that what truly matters is not just shipping those outputs, but rather what outcomes they drive.

What does a culture of experimentation mean?

In today's competitive business environment, a culture of experimentation is not optional for product teams. The work teams do must align with both user and business value. A culture of experimentation helps you to de-risk projects and deliver value by breaking down huge organizational goals into smaller, testable hypotheses that a squad can own. Organizations that ignore this approach will find themselves going down costly dead ends and falling behind.

A culture of experimentation must permeate every level of an organization, with a particular focus at the squad level. Squads are the atomic unit team that solves a customer problem, usually: product, design, and development. Some squads benefit from additional partners like research, marketing, and others. This grassroots approach fosters autonomy: empowering teams to generate hypotheses, design experiments, and learn from their outcomes.

As experimentation spreads to other squads, apply lessons learned to new teams and unify the culture across the org. It's critical to have an experimentation strategy here, and to avoid a scattershot approach of running a thousand small A/B tests whose learnings do not compound. A culture of experimenting at the team level creates a powerful foundation for agility and adaptability. This allows companies to conquer challenges and seize opportunities in an ever-changing landscape.

Why the continual mention of culture? People may think: "Teams need to experiment. Platforms such as Amplitude Experiment support that need." It is a powerful product, but it's only part of the solution because culture is not purely technical. Ultimately, culture is a product of people, integrated processes and communication, and technology. Let's explore these three elements in more detail:

- **People.** It starts with resourcing an empowered squad. Across the squad, members should be data-driven and empowered to own and solve the most important problems facing customers and our business.
- **Integrated processes & communication.** Experimentation must be integrated into the product development process from the outset, with healthy communication across teams. Hypotheses, test design, success metrics, and implementation must be considered early—not in the final stages of a launch and certainly not ad hoc after launch. Open communication to celebrate wins and losses, share customer insights, discuss ideas, and plan for the future fosters a strong culture of experimentation.
- **Technology.** Complement the right people and processes with a robust platform that enables the best work to be done. The technology you use should not only measure product analytics but also include a framework for A/B testing.

Building a culture of experimentation

We have an understanding of the foundational components of strong experimentation culture—people, process, technology, and communication—however, we would be remiss not to mention other essentials as you work to build a culture of experimentation at your organization.

- **Secure an executive champion.** An executive champion should help generate momentum, support risk-taking, and encourage learning from failures.



- **Be early and rigorous in your hypothesis design.** Experimentation shouldn't be an afterthought. Define hypotheses and success metrics when a project starts; don't try to figure these out as you launch.
- **Embrace failure.** To truly build a culture of experimentation, organizations must embrace failure as a mechanism to learn. If you never fail, you probably are not pushing hard enough on new ideas.
- **Break grand challenges into smaller testable hypotheses.** Embracing failure works when individual components of a feature are tested and invalidated versus the feature itself. Don't ship a whole product and see how it goes. Instead, de-risk the development of large features by testing smaller components. This allows you to figure out which components are validated by customer data and determine how to iterate. After all, the faster you drive the feedback loop of shipping to customers, measuring, and iterating, the faster you succeed.
- **Establish one center of excellence.** Start small and focused with one team to validate the basic experimentation stack and build from there.

“ A culture of experimentation is not solely a technical problem to solve. Culture is a product of people, integrated processes and communication, and technology.



- **Cultivate curiosity.** As previously mentioned, the goal is to maximize outcomes and minimize the outputs it takes to get there. To build a culture of experimentation, you must get management addicted to outcomes. Start with projects your team has already shipped. This should be easier to measure, exposes gaps in measurement ability, and kickstarts the mindset that shipping means we get outcomes.
- **Build a high-trust experimentation system, even if only measuring simple metrics.** Waiting to measure long-term results can negatively hurt momentum. Establish proxy metrics that help shorten the lifecycle of the experiment. Make sure your proxies are upstream and can inform whether you're going in the right direction or not.

All of these elements are important to building a sustainable culture of experimentation, but I want to dive a little deeper into the role executives play in enabling a culture of experimentation.

Executive support is important for building a culture of experimentation because it helps generate momentum and encourages risk-taking and learning from failures. Other ways executives champion your budding experimentation culture include:

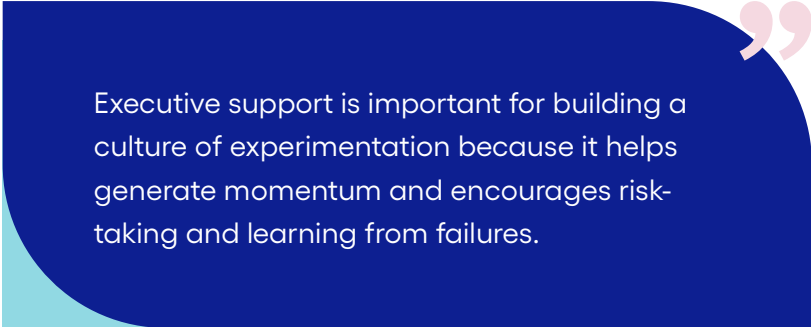
- Providing resources, such as time, budget, and tools, to support experimentation and empower teams to take action.
- Celebrating and recognizing successes and failures, as both provide valuable learning opportunities and help build a culture of continuous improvement.

- Creating channels for open communication and collaboration across teams and departments to facilitate sharing of knowledge, insights, and best practices.

One thing championing executives don't do is request laundry lists of features to ship without connection to strategy or goals. Doing so reduces product empowerment, squashes any bandwidth for experimentation, and hurts teams in the long run.

Cultural barriers to experimentation

At Reforge, we care so much about experimentation that we built an entire program around creating and executing a strategic experimentation system for breakthrough ideas. The program starts with experts Elena Verna and Fared Mosavat breaking down cultural barriers to experimentation. [Apply to Reforge for access.](#)



Executive support is important for building a culture of experimentation because it helps generate momentum and encourages risk-taking and learning from failures.

The Importance of Experimentation and How to Get Started



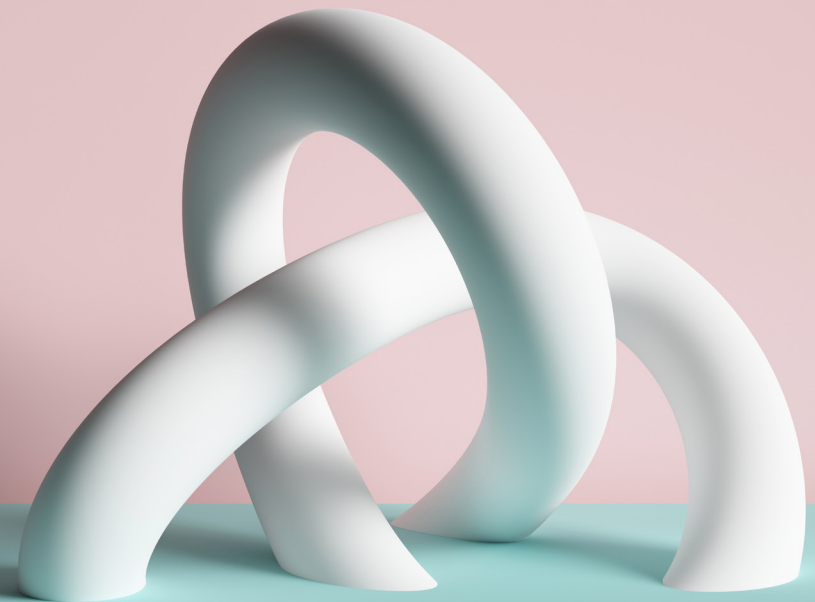
By Bhavik Patel
CAUSL

As businesses move from output-based to outcome-based approaches, product leaders—and their investments—are facing greater scrutiny. I saw this growing trend firsthand in my work heading analytics and experimentation for startups such as MOO, PhotoBox, and Hopin. In fact, it's why I founded my product measurement consultancy CAUSL.

This shift in accountability has led to something of a heyday for experimentation. There's a good reason for that.

Experimentation—specifically, A/B testing—is the gold standard when it comes to making decisions about iterating on features or products, or launching entirely new ones. It also makes it possible to quickly assess the impact of those product decisions before committing to them. And that's important because launching features and products is expensive: the total cost goes beyond deployment to include research, design, engineering, data and analytics, and more. In other words, it's a way for businesses to “de-risk” their investments.

Experimentation—specifically, A/B testing—is the gold standard when it comes to making decisions about iterating on features or products, or launching entirely new ones.



But there's another reason product teams are turning to experimentation. While measuring the ROI for something like marketing is relatively straightforward, that's typically not the case with product. After all, product investments are measured by things like headcount and engineering hours. And while product teams hope they can tie spend to company goals like revenue growth, customer growth, or conversion rates, there are only a few ways to do that. Experimentation is one of them.

The importance of good experimentation design

Experimentation is important—but it's only useful if it's done right. What does that mean? Well-designed experiments are rooted in the scientific method and can be broken down into the following phases and steps:

Planning (pre-experiment)

- Document the reasons for running the experiment.
- Conduct research to validate your observations. This research can be quantitative, qualitative, competitive insights, and more.
- Build a hypothesis with clearly defined product metrics or KPIs.
- Create alignment on next steps based on each possible outcome: whether each variant wins, or if the test does not reach statistical significance.
- Other key considerations include the test duration, audience if it is a targeted experiment, traffic split, critical threshold, and more

- Clearly articulate what is being tested and modified. This could be a design change, a new feature or experience, and more.
- Confirm that the experiment can be measured using analytics tools.

During the experiment

- Create analytics dashboards for ongoing monitoring.

Post-experiment

- Analyze the results.
- Articulate the outcome of the experiment and move forward with the predetermined next steps.

Building vs. buying an experimentation platform

Of course, any discussion on how to run experiments inevitably leads to the question of where to run them. In other words, is it best to build an experimentation platform or buy one? My answer is always a resounding “buy.” In my view, experimentation is a commodity. Most companies wouldn't build a CRM platform, so why should they build one for experimentation?

Experimentation platforms are the product of numerous statisticians and engineers who have ensured the platform meets statistical requirements on elements such as randomization and statistical engines. They also come with user-friendly interfaces making the platform accessible to different team members. Building something

similar would require two or three years, plus ongoing maintenance—and even then, it might result in running experiments with built-in biases that lead to product teams making poor decisions.

Measuring A/B testing success

Like any good program of work, A/B testing demands its own method for assessing success. In my view, this comes down to: velocity, process, and win rates.

Velocity

Companies just starting with experimentation should look at the number of tests run, quarter over quarter, year over year. Numbers aren't everything, but they can serve as a good indicator of whether a team or company has built a culture of experimentation.

Process

As the experimentation program evolves, the focus should shift to the quality of experiments and the process they follow to go live. That means ensuring the process adheres to scientific methods—in particular, grounding hypotheses in evidence-based research, whether through quantitative data or qualitative data such as customer feedback or service tickets. It also means that there is an optimized and documented flow in which an experiment can go from an idea into production that accounts for prioritization, development time, experiment length, and expected impact.

Win rates

Ultimately, every team running experiments will also want to evaluate its win rate. While it's natural for that number to fluctuate at the outset of an experimentation program, it should stabilize over time. As a rule, experimentation programs should seek an approximate win rate of at least 1 in 5. If your win rate is lower, you may have a problem at the hypothesis and research stage. But it's not quite that simple. While those wins represent value, the losses are worth something, too. They are the bad decisions that were rolled back, ensuring the risks—and costs—were averted. Even tests without a clear winner can help save on product costs and feature bloat.

As a rule, experimentation programs should seek an approximate win rate of at least 1 in 5.

Experimentation milestones

As experimentation programs mature, there are a number of other milestones to keep in mind, including growth in the number of teams running experiments. After all, experimentation is something to democratize—it shouldn't belong to one team. The entire organization should embrace testing.



Reliable tooling is another important goal post. If teams can't trust their instrumentation and tooling, they will inherently distrust their results and question the experiment's validity. Ensuring your teams have a reliable stack is essential to build trust across your organization.

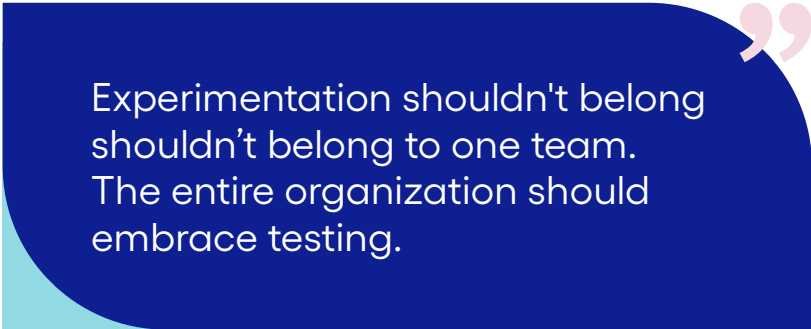
Another key milestone is buy-in at all levels of an organization, including the top. In the same way that reliable instrumentation builds trust in testing, a business-wide culture of experimentation means stakeholders are a lot more likely to align on outcomes—freeing them to focus on what to do next.

Best practices for launching an experimentation program

Newcomers to experimentation should start by building their testing muscle memory by establishing a regular cadence for experiments. There's no need to get overly academic at first. After all, experimentation itself is trial and error, and that's what makes it such a beautiful concept to adopt. It's about finding what works, not nailing it out of the gate.

To do that, it makes sense to start with low-risk tests. I wouldn't advise starting with a complex pricing experiment, for example, but it's a great goal to work toward. Another pitfall to avoid: getting hung up on arbitrary—and quite often unrealistic—numbers. Companies often assume they should be running thousands of experiments because they compare themselves to Amazon or Netflix, but reaching that

scale requires an enormous amount of traffic and resources. For most companies, the goal shouldn't be running thousands of experiments; the goal should be running more than they did last quarter or year.



Experimentation shouldn't belong
shouldn't belong to one team.
The entire organization should
embrace testing.

Make better product decisions

These are just a few of the easy ways to get started on an experimentation program, and there is little reason to wait. A well-designed, science-backed experimentation program is one of the most powerful tools available for making smarter product decisions—from when and what to launch to measuring the ROI. And with product teams under ever greater pressure to spend wisely, that's more important than ever.

Experiment Ideation and Roadmap Development



By Bhavik Patel
CAUSL

Process matters when it comes to experimentation. And that goes for experimentation ideation, too.

Knowing what and where to test is mission-critical to both the experiment and the experimentation program. Ad hoc or misguided testing can shake an organization's faith in the entire program, which leads to bad decision-making and undermines data-driven cultures.

To truly understand every element of a platform or product and how they ladder up to an organization's business goals, I like to create a "metric tree." Metric trees chart how various metrics from features and products relate to one another and the business' growth levers. After all, those are the same levers you want your experiments to optimize. Aside from helping you understand how your business makes money, metric trees can help compartmentalize experiments and give you an idea of where to focus your efforts.

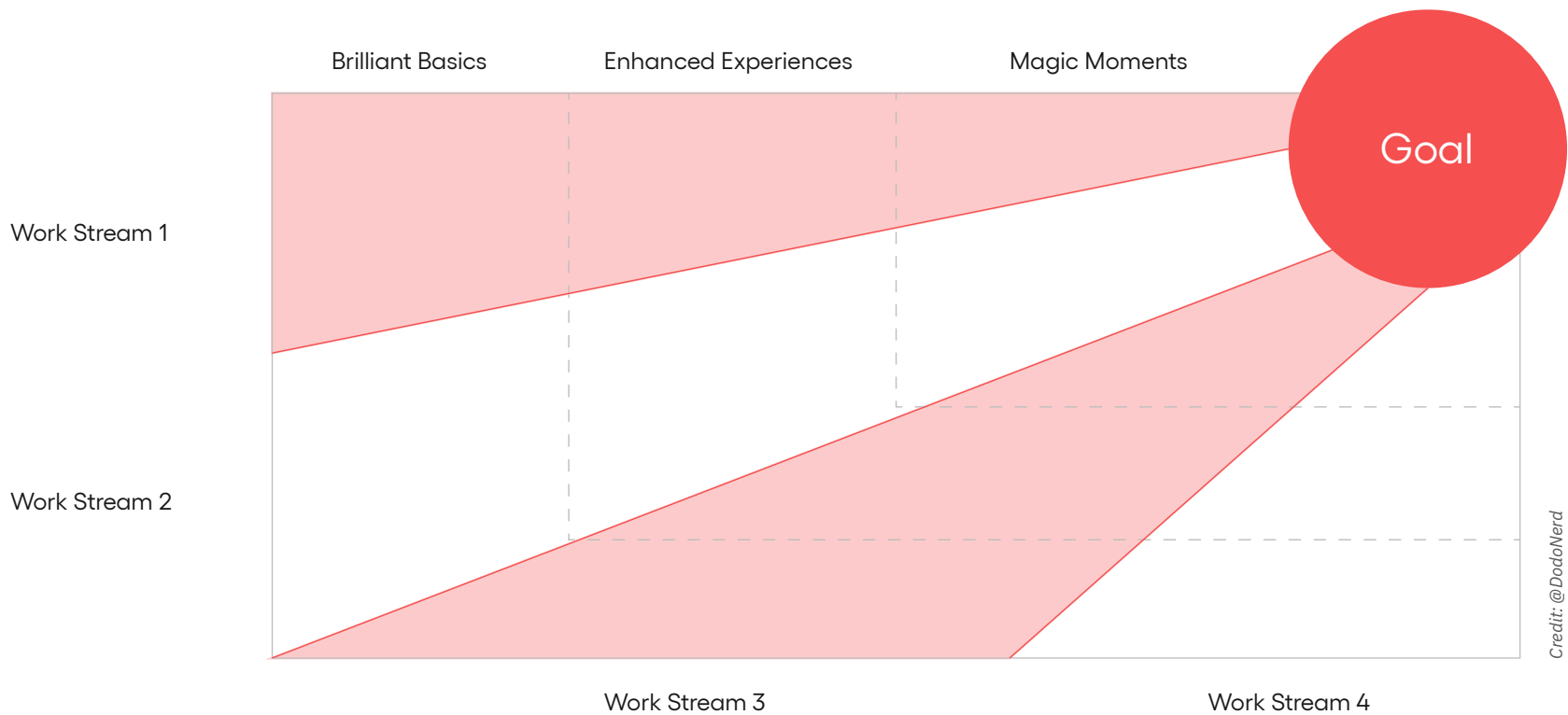


Credit: @DodoNerd



[Sun Diagrams](#) are another useful tool for ideation. In this diagram, the sun takes the form of a goal, and the sun's rays are workstreams. At the end of each workstream is a team or pod that is organized according to, for example, the Pirate framework for growth. (For the uninitiated, the framework focuses on the customer acquisition stages of acquisition, activation, retention, revenue, and referral.) I like to divide the diagram into three columns to sub-categorize the workstreams by experiment type: "brilliant basics," "enhanced experiences," and "magic moments."

Brilliant basics cover experiments to meet customer expectations, such as signups and onboarding flows. Enhanced experiences refer to intuitive experiences like search experiences and wish lists that allow customers to discover and use the core product. And finally, magic moments are the kind of experiences and innovations, like recommendation engines, that differentiate the product and delight customers.



Credit: @DodoNerd



How to validate experimentation concepts

Experimentation can be a big cost-saver; every “failed” experiment represents features and products you shouldn’t have launched. But experimentation is still an investment, potentially requiring significant engineering hours. This is why it’s helpful to have a method for determining which experiments are worth pursuing and which are not. Fortunately, there are a number of techniques to help guide your experimentation roadmap.

Painted door test

As its name suggests, a painted or fake door test offers a button or link to gauge customer interest in a new feature or product. Of course, when customers click on those features, they discover that feature isn’t available, often finding a form to “learn more.” This can lead to unintended consequences such as disappointing and, in rare circumstances, even alienating users. But it does provide great data and insight for further investigation.

Sign-ups

Another easy way to measure interest is to ask customers to sign up to stay informed about whatever feature you’re building. You can do this through your website or social media accounts.

Desk research

Desk research involves looking into whether other companies have published studies about tests they have done on the same sort of feature you’re creating. This can save you time by learning from the

experiences of teams at other organizations and help you understand what uplift they were able to generate based on their learning.

Similar experiments

Looking at previous experiments of the same size and product area can serve as a useful guide or proxy when it comes to assessing potential uplift, adoption, or usage.

Leap of faith

Sometimes, the only way to validate an experiment is to take a leap of faith and push forward with it. After all, validating experiments requires experimenting. Remember that A/B testing is a great tool for decision-making, but it shouldn’t hinder your ability to make a decision. Sometimes, you just have to roll the dice.

When not to run an A/B test

In some situations, A/B testing may not be appropriate or even possible. Pricing is a prime example of an area where experimentation may be ethically untenable. It can also cause customers to complain—potentially very publicly—that they are being charged more for the same goods or services than other customers.

Fortunately, there are alternatives for these scenarios, with analytics playing a key role in each approach. Taking the pricing example again, one option is a geographic-based study of customers in two similar markets. Observing the effect higher pricing has in one market can indicate the impact of higher prices across the board. Estimating the causal impact using regression modeling is another useful approach—it

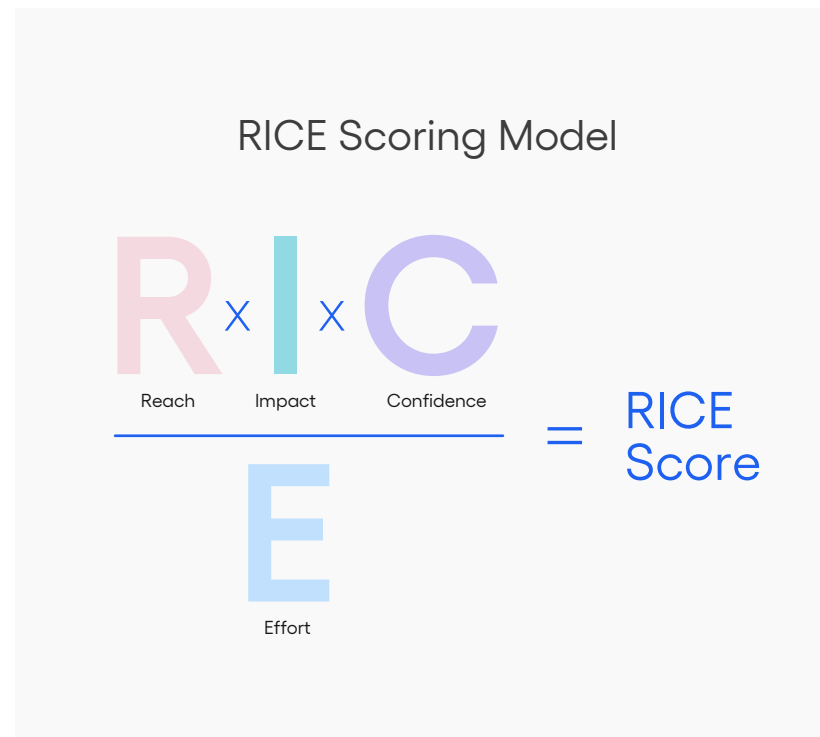
looks at how an input affects a particular metric and how a change in that input affects the metric. Pre-post analysis also avoids classical testing by comparing data gathered before and after a change.

Some teams may even roll out an experience or feature without running a test. Of course, if it “flops,” you can always roll back the experience, but keep in mind that by not experimenting, you won’t be able to say with confidence that the downturn of your metric was due to the feature or some other external factor. The same is true for the opposite scenario. This practice may be especially tempting when your team is confident about the expected outcome. But even if your intuition is correct, you’ll miss key data around the magnitude of the impact, which is critical to demonstrating the impact of experimentation.

There’s something else to consider with this practice. Rolling out features without testing in the early days of a startup makes sense, but after a certain point, if every feature is released without validation, you’re setting a dangerous precedent that output is preferred over outcome. In my opinion, this leads to feature bloat. Experimentation is a healthy way to balance intuition-based decisions and evidence-based decisions.

How to prioritize experiments

Popular tools such as the RICE scoring model are very effective for determining where experiments belong on a team’s roadmap. The RICE scoring model measures initiatives according to their reach, impact, confidence, and effort.



” Rolling out features without testing in the early days of a startup makes sense, but after a certain point, if every feature is released without validation, you’re setting a dangerous precedent that output is preferred over outcome.

That said, RICE scores are not the only factors to weigh when prioritizing testing. Most companies will need to balance them against the availability of traffic and internal engineering and design resources. Often it’s a question of whether to tackle one big test that takes up all the organization’s bandwidth or taking on a number of smaller tests. These are the tradeoffs most businesses have to make, especially if they don’t have a large volume of traffic. And let’s face it, that’s most companies.

Build a lasting culture of experimentation

Choosing the right experiments—and when it’s better to take another route—may require some time and effort. But putting in the work of thoughtfully ideating, validating, and prioritizing them is well worth it. It will pay off in the form of an experimentation program embraced by your entire organization and a strong data-centric culture.



Product-Led Experimentation for B2B Organizations



By Elena Verna

[Product-led growth \(PLG\)](#) has provided B2B product teams an opportunity to drive meaningful impact at their organizations, but it changes how product managers should view their position. As organizations invest in PLG initiatives, they inevitably put more accountability on the product to deliver growth outcomes, including revenue.

This is a fundamentally new motion for B2B product teams since growth has primarily been driven by sales and marketing, not product. In sales-led motions, product teams are measured on release velocity and the number of features shipped each quarter. As a result, they invest in tools and processes focused on speed to delivery rather than direct impact on business KPIs.

In PLG motions, simply shipping fast creates substantial exposure to the business, so what happens if you ship the wrong feature or experience? Key KPIs like user growth, subscription revenue, and retention can all be negatively impacted.

But if product leaders shift their approach, tooling, and mindset to tie experimentation to the development process, product teams can deliver high-impact releases that directly influence revenue as they continue to adopt PLG.

In PLG motions, simply shipping fast creates substantial exposure to the business.

The dangers of not experimenting

Consider the implications of continuing to develop your product without experimentation in a PLG motion. Without a clear, data-driven approach to product development, all decisions will revert to the opinion of the HIPPO (also known as the “highest paid person in the room”) or be based on intuition. This kind of decision-making is not good for your product or your customers. Even if you stumble into a positive change, your success will not be repeatable.

Intuitive decisions sometimes work. It makes sense that someone who knows a product and user base well will be able to instinctively know what changes to make. However, when your product and market changes, your intuition expires.

In most products, there’s a perception and reality gap between what customers need and what you think they want. As your B2B product grows, that gap also grows. You need to regularly update your knowledge by gathering data through experimentation.



There's more to experimentation than A/B tests

Experimentation doesn't mean running A/B tests across your whole product. There are several types of tests product development teams can run to learn about their users and their product.

Even with A/B tests, you can run a full release or a partial rollout across a small section of users. You can also run:

- **Pilots and beta features:** A group of users tries your product or feature and gives feedback before it's ready for general release.
- **Wizard of Oz tests:** You create a mock interface you control, allowing users to try out a product before you build it.
- **A painted door:** Check demand for a feature by creating a fake button or CTA with a placeholder to say the feature isn't available yet and track the number of clicks.

Another option is to ship small changes and analyze the data pre and post-release.

The experimentation journey

A culture of experimentation doesn't magically appear overnight. A single team needs to first learn how to test, how to learn, and how to win. Once that team has developed an experimentation skillset and mindset, they should work on democratizing access to testing and learning. That way, the entire organization can learn how to win with experimentation.

How to experiment in B2B

Based on my experience, here are the steps for successful experimentation in B2B organizations.

Prioritize the velocity of learning

The velocity of learning is the [North Star Metric](#) of experimentation. To learn quickly, you need to experiment frequently with tests that deliver learnings within a short period.

Teams should start by producing a breakdown of their assumptions about the product. Then, they have to create new development lifecycle processes focused on data collection about assumptions rather than full feature availability.

The correct tooling to run tests, data efficacy, and a culture of forgiveness rather than permission are also essential to a high learning velocity.



Shift your mindset

Organizations usually start with the wrong mindset: they want to see clear wins from experimentation. They tend to only treat an experiment as successful if it brings a lift in a metric they want to improve.

But tests that “fail” because they identify a product variation that underperforms or doesn’t deliver significant movement are more valuable than a metric lift. These “failures” are useful because they give teams a concrete example of what not to do. A shift in your mindset and definition of experimentation success is critical to driving learning velocity and promoting healthy accountability within your organization.

Identify where you have enough volume to test

A common problem for B2B organizations is they don’t have enough traffic, leading to weeks or months before they see test results. But if you can’t observe meaningful movement in two to three weeks, your velocity of learning will drastically decelerate.

Imagine that to run a certain test you have to spend eight months on the experimentation cycle—from ideation to development to testing. You’ve now spent an enormous amount of time and resources on only one set of learnings.

Map out your customer journey to identify areas with enough traffic to test based on the minimum detectable effect (MDE) you want to drive. If you want to prove a smaller MDE, you need much higher traffic than trying to validate a larger MDE. While this seems counterintuitive, this



insight is important to help you define what tests you should consider based on the desired impact on your key metrics.

If you have a product-led growth motion, you will likely find that your dataset looks similar to B2C companies, with more traffic at the top of the funnel. Your homepage and landing pages would be ideal high-traffic places to start running A/B tests.

If you have low-traffic areas you want to test, you can take other data-driven approaches, such as the different testing methods we listed above (pilots, beta features, etc.). These tactics are much easier to execute while still providing great insights and data for analysis.

AMPLITUDE CASE STUDY

Self-serve checkout volume

At Amplitude, we recently launched a self-serve checkout. There wasn't enough volume to run standard A/B tests, so we opted to test with a pilot targeted at a segment of customers. We then vigorously analyzed the impact pre vs. post-release.

Once we get more volume in our self-serve monetization experience, there may be an opportunity to A/B tests in the future. And since there's more volume in other areas—like the pricing page, upgrade triggers, and dashboard view—we prioritize A/B testing these learning opportunities instead.

Don't test for the sake of testing

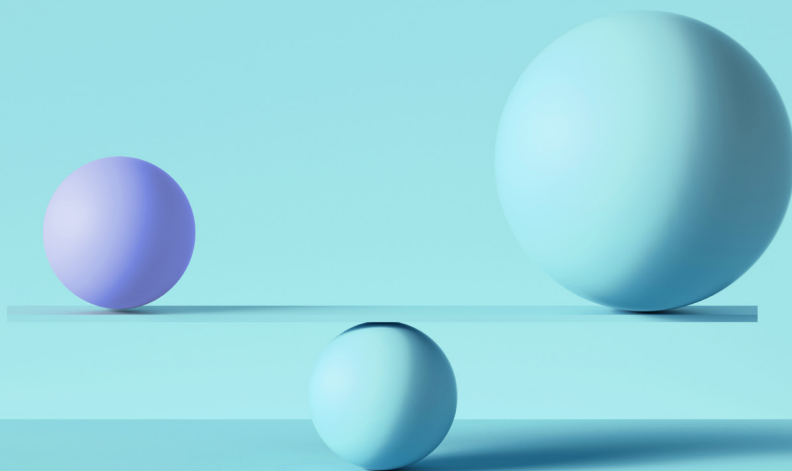
Not everything should be up for experimentation. It is unsustainable for your team to run tests in every product area.

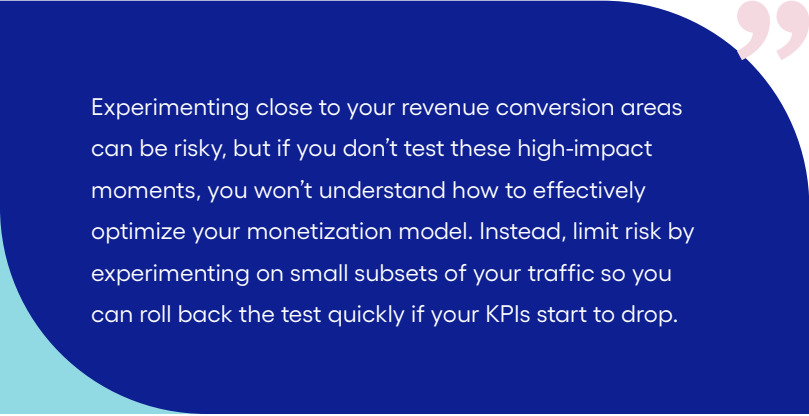
Only experiment when:

- **Your test hypothesis aligns with the business strategy.**
Your test needs to relate to the lever your org is focused on influencing.
- **You have enough traffic.** If there's not enough volume, it'll take a long time to show results.
- **Your test data will inform decision-making.** If you don't need precise quantitative data to decide, a test isn't necessary.
- **The cost of running a test is lower than its potential impact.**

Let's say you want to test including an extra email in your welcome flow to drive retention. There's enough traffic, which means you could test it. But do you really need to know the impact of an additional email on the business to the last cent? You're better off testing a bigger bet and saving your team's capacity to focus on a potentially more impactful experiment, like checking how users respond to a new feature.

Prioritize tests based on the cost of the test, its potential impact, and your confidence that it will have an impact.





Experimenting close to your revenue conversion areas can be risky, but if you don't test these high-impact moments, you won't understand how to effectively optimize your monetization model. Instead, limit risk by experimenting on small subsets of your traffic so you can roll back the test quickly if your KPIs start to drop.

Roll out experimentation across your organization

Democratizing testing allows you to increase the velocity of learning across your entire organization. However, democratizing too soon can create chaos and data drift. Start by creating experimentation systems on one team before rolling out testing across the organization.

When you're ready to expand experimentation, focus on making testing more efficient. Improve your processes and tooling to reduce the costs associated with each experiment. This increases the number of tests you can run across members and teams within your organization.

As you scale experimentation, teams also need permission to experiment in multiple product areas. People need the ability to pull a lever where it matters most. When engineers want to test but aren't allowed to write code in the surface area their hypothesis impacts, they waste resources chasing tests that don't deliver learnings.

Another situation that limits experimentation is when people are too scared to test things related to their monetization model—like feature allocation or prices. In this case, they restrict testing to less critical areas like acquisition.

Experimenting close to your revenue conversion areas can be risky, but if you don't test these high-impact moments, you won't understand how to effectively optimize your monetization model. Instead, limit risk by experimenting on small subsets of your traffic so you can roll back the test quickly if your KPIs start to drop.

Make sure testing doesn't paralyze decision-making

While it's true you should prioritize the velocity of learning by increasing testing across your organization, there's a limit. Too many data points can slow you down.

When I worked with SurveyMonkey, we did a good job of increasing our velocity of learning by developing a democratized culture of experimentation. But then we hit an unexpected challenge: no one wanted to release any feature without testing it first. Our team fell victim to "analysis paralysis" since tests became too much of a crutch and slowed down our innovation cycles.

To avoid analysis paralysis, ruthlessly prioritize the tests you run. First, map the average traffic throughout each part of your product, then assess your engineering capacity to help build, QA, and deliver each

test. Next, use this information to identify how many tests you can run and compare it to how many tests you are currently running to see if you are under or over capacity.

If you are under capacity and can test more, empower more teams to start experimenting in different product areas. If you are testing at capacity but not learning much from your tests, consider running different tests and reprioritizing your experimentation roadmap. If you are over capacity, be more selective about the tests you run; more testing isn't always better.

Experimentation metrics

Each test should focus on one revenue-related lever; the remaining levers become your guardrail metrics. Organize your metrics into a [data-hierarchy map](#).

In B2B, there are four main levers with [KPIs that influence revenue](#):

- **Acquisition metrics:** prospecting traffic and new sign ups
- **Activation metrics:** set up, “aha” moment, habit-forming moment
- **Engagement metrics:** frequency of use: power, core, casual
- **Monetization metrics:** free-to-paid conversion, expansion, renewal

For each experiment you run, select one of the four levers as the main success metric for the test. Track the remaining three as your guardrails to de-risk your experiment. For instance, if you're running

a test related to acquisition, set up a dashboard to monitor activation, engagement, and monetization metrics.

If your test starts to harm these guardrail metrics, you will receive rapid feedback, giving you a clear signal to roll back the test.

If your test starts to harm these guardrail metrics, you will receive rapid feedback, giving you a clear signal to roll back the test. With long-running experiments, you can even monitor the impact of your test for months after the test period is over since there could be latent effects.

As you build experiments and focus on guardrail metrics, be sure to track the guardrails in time cohorts—for instance, what happened over 24 hours versus over one week. This step will help you identify if you have a [pull-forward effect](#) or latent impact compared with tracking a running total.

Create a culture of learning with experimentation as your third data set

When considering the most important data for their business, companies typically think about two primary datasets:

- **Qualitative data**, such as user needs and motivations captured by user interviews.
- **Quantitative data**, such as transactional production data, behavioral data, or third-party recorded data, such as CRM.

When teams only work with qualitative and quantitative datasets, they tend to observe [correlations](#) between data points: action A relates to action B in some way. But if you want to impact business, you need to identify causal relationships—where action A causes outcome B. Causal relationships tell you that if you move a certain lever, you'll get a certain outcome.

B2B organizations that invest in experimentation create a third dataset from their experiment results, enabling them to identify causative relationships. Plus, it informs your experimentation roadmap since you can review results from previous experiments to accelerate learning without harming your capacity to test.

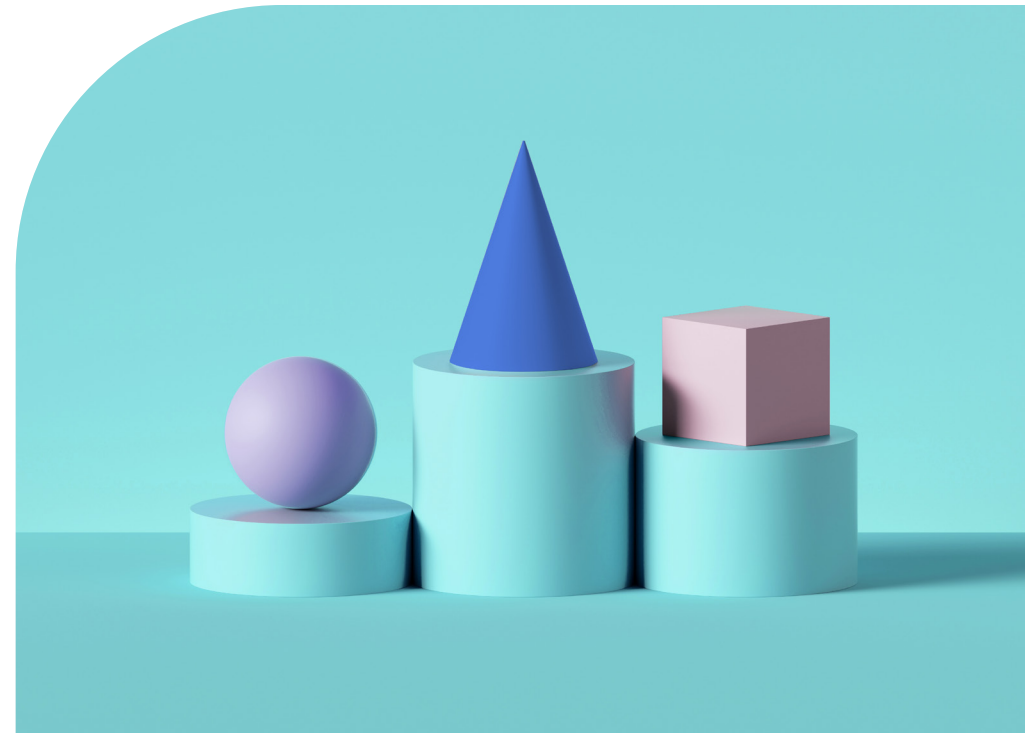
To build your experimentation data set, create a repository of every test you've run with clearly articulated results. Make the library accessible to everyone, so people who work across all parts of your product can add to and learn from it.

Enforce rituals around experiment results. Share experimentation learnings with the whole organization (for example, at All Hands meetings) in the same way as you'd share results from user interviews. Include every test, regardless of whether they were “successful” or not.

Get started with an experimentation framework

To help you bring more value to your B2B organization with a sustainable experimentation program, I created a [seven-step experimentation framework](#). Use it to ensure your efforts are aligned with business growth and customer problems.

To dive into product experimentation further, check out my [Experimentation + Testing Reforge program](#).



Feature Management and Experimentation: Two Sides of the Same Coin



By Wil Pong
Amplitude

In recent years, big companies like Netflix changed the game when it came to offering personalized user experiences.

The streaming platform provides users with tailored viewing recommendations based on viewing history, search query, and rating data. Instead of just showing one set of recommendations, Netflix also uses A/B testing to experiment with different suggestions and interfaces. With this experimentation, the platform can iterate and refine its platform in real time based on user preferences.

Until recently, this sort of work was only possible in large companies that could invest in machine learning technology and teams of data scientists running thousands of experiments every year. But this is no longer the case. Third-party applications like [Amplitude Experiment](#) put the power of feature management and experimentation into the hands of organizations of all sizes.

Third-party applications like Amplitude Experiment put the power of feature management and experimentation into the hands of organizations of all sizes.



As the head of experiment products at Amplitude, I have worked with data pioneers like Netflix, Uber, LinkedIn, and Microsoft. I specialize in democratizing advanced consumer-type techniques, such as feature delivery, for companies of all sizes. In this chapter, I will discuss the relationship between experimentation and feature management and how these help teams deliver personalized experiences.

How does feature management relate to experimentation?

Software's big advantage over other products is that you can instantly change it. When you build a PS5 controller and ship it off, you can't change the controller in that box once it leaves the factory. It's already out there. We can change software as fast as we have ideas.

Through feature management, we can use techniques like [feature flags](#) that allow you to toggle on and off the specific features you want your users to experience. You can also toggle different features for specific user groups, or cohorts, based on their behavior or other characteristics.

Using feature flags allows teams to test features in a production environment. Normally, new features are released to specific cohorts, not the entire user base. This way, development teams can gather user feedback before fully releasing new features. When teams deliver A/B tests, they go through this same process, which explains why technical teams view experimentation and feature management as synonymous.

How does your experimentation system make targeting possible?

It is critically important for your experimentation platform to have [identity resolution](#) built into it—meaning the solution can identify the same user even if they're logging in from different devices or contexts. Without identity resolution, you risk misidentifying your users, leading to flawed experimentation and bad data. You could identify a power user as a new user just because they've logged in through a new device.

If identity resolution isn't built into your platform, you will need to constantly ping other systems, like a CDP, to double-check your users' identities. This will introduce more latency or lag to your system. You will also incur the high costs of building integrations and managing data pipelines between your platform and something like a CDP.

When you combine feature management capabilities with accurate identity resolution, you suddenly have the ability to deliver highly targeted experiences to the right cohorts of users. For product and engineering teams, this is the pinnacle of A/B testing.

When you can reliably harmonize user identity in your evaluation engine, you can create rules based on specific user attributes, like past behavior and geographic location. These rules give you control over who gets what features and when.

For example, you can carry out an experiment where only your power users—people who log into your product more than three times a week

and spend some minimum amount—get access to new deployments. You want to test a new feature with your most valuable users to see how they will react.

This type of targeting is only possible when you have an evaluation system such as Amplitude Experiment that tracks the information needed to segment these cohorts. If your experimentation system does not have connections to this information, you'll need to spend significant resources connecting to relevant data sources, like CDPs or data warehouses.

On the other hand, testing and deploying on a code level allows you to deliver a much wider range of experiences, whether they are short-term or long-term updates.

How does feature management differ from a “WYSIWYG” approach?

The feature management approach to testing and deploying new features is made directly at the code level. Other approaches to deployment allow changes to be made on a surface level without altering code. One such approach is through “What You See Is What You Get” (WYSIWYG) user interfaces such as WordPress for websites

or Appy Pie for apps. These interfaces allow companies to make deployments and experiment without using code.

However, using WYSIWYG for deployment has significant limitations.

WYSIWYG only works in limited cases

WYSIWYG gives you a small subset of things you can alter and only works when you want to roll out changes for a fixed period. For example, your growth and marketing teams may need to launch holiday promotion coupons to new users through push notifications. This example is an ideal use case for WYSIWYG A/B testing. You can predominantly change colors, copy, or labels, which is why marketing teams have historically used it on landing pages or website copy changes.

On the other hand, testing and deploying on a code level allows you to deliver a much wider range of experiences, whether they are short-term or long-term updates. An ecommerce company, for example, can use feature flags to stack rank its best products or new promotions on screen or run tests on their onboarding experience more holistically. Since you are making changes at the code level, product and engineering teams can have far more control and flexibility on the experiments and tests they choose to prioritize.

WYSIWYG-based tests may deliver actual value, but marketing teams will have to wait for product and engineering teams to have the capacity to deliver a long-term update.

WYSIWYG is fast in the short term but slow in the long term

WYSIWYG is not built into the flow of rapid product innovation like CI/CD processes. Instead, these approaches lead to parallel processes for experimentation and product development. WYSIWYG-based tests may deliver actual value, but marketing teams will have to wait for product and engineering teams to have the capacity to deliver a long-term update, interrupting their typical workflow to add new capabilities to their product. As you can see, WYSIWYG is designed for experimentation with static or incremental changes, not complex or dynamic experimentation scenarios.

By using feature flags, engineering teams can build tests directly into their CI/CD processes, allowing them to make the necessary changes to backend systems and making it easy to roll out the winner once the test is complete. When combined with analytics, feature management also allows developers to get more data about every feature they release. This provides them valuable insight into their work's impact on the broader organization—something that WYSIWYG tools will never provide.



WYSIWYG slows down the performance of your application or web page

When you use WYSIWYG, you're overlaying a website on top of your main website, which can negatively affect loading times—sometimes doubling or even tripling it. This overlaying can also lead to the flicker phenomenon—when a web page or application's user interface changes or reloads, causing a visual disturbance or glitch that can distract users. This happens because changes to the visual interface are not immediately reflected in the code.

There has also been considerable research highlighting that any impact on performance has a significant impact on conversion rates. At Amazon, a 100-millisecond slowdown experiment [decreased sales by 1%](#). This stat helps to highlight why product and engineering teams need to be mindful of their experimentation solution's performance.

Although deploying features through WYSIWYG is much easier, it is ultimately not as extensive, effective, or efficient as using a feature management approach.

Product experimentation delivers faster innovation and ROI without disrupting development cycles

As the Netflix's of the world have shown, product and engineering teams have a huge opportunity to deliver more ROI to their

organizations through personalized experiences—even if you don't have millions of users visiting or engaging with your product every day.

If your experiment platform has a robust approach to identity resolution and delivery, product teams can deliver highly targeted experiments and feature releases using the same processes and technology that power the rest of your product development. This will ultimately drive rapid innovation without disrupting technical teams.

If your experiment platform has a robust approach to identity resolution and delivery, product teams can deliver highly targeted experiments and feature releases using the same processes and technology that power the rest of your product development.

Critical Capabilities for Product and Engineering Teams to Scale Experimentation Programs

Product and engineering teams can stumble trying to build a data-driven culture when they focus too much on technology decisions before defining their specific business requirements.

Once business objectives are defined, then product organizations should start looking for the essential capabilities they need to scale their experimentation platform.

This section provides a baseline of functional capabilities that product and engineering teams should consider when investing in product-led experimentation.

Guide teams to deliver trustworthy experiments every time

Experiment design is a critical part of any A/B test. Teams new to A/B testing can suffer major data quality issues due to poor design and implementation. Best-in-class tools reflect testing best practices, helping teams deliver high-quality tests and make optimal product decisions.

Another important consideration is the variety of testing capabilities your team needs to succeed and scale their programs. Mutual exclusion groups ensure teams can better manage simultaneous tests by restricting traffic across colliding tests. Holdout groups ensure that some traffic is restricted from ever being exposed to experiments, providing a new way for teams to baseline their tests.

KEY QUESTIONS

- Does the solution adhere to A/B testing best practices and key testing milestones like hypothesis development, targeting, delivery, allocation, monitoring, and analysis?
- Can I easily understand how much traffic and how long my test will need to run within the UI?
- Can I seamlessly access my key metrics for my A/B tests?

Trustworthy statistical rigor and support for your most important use cases

Understanding the statistical rigor of your experimentation platform is another important consideration for product experimentation. While there is no “right” answer regarding the “best” statistical methodology, each approach does have its tradeoffs and advantages. Understanding your experiment platform’s approach is important in order to have the utmost confidence in your results.

Some statistical methodologies, like Sequential Testing, allow product teams to peek at results midstream without impacting their results. Whereas T-Tests are more popular, but require teams to wait until the test concludes to see their results - otherwise they risk degrading their results. While these differences may appear trivial, they can have profound implications for experimentation programs.

Another important consideration is the types of testing use cases your team needs to succeed. Teams can choose from a wide variety of use cases, including A/B tests, multivariate testing, multi-armed bandits, do no harm tests, quasi-experimentation, and more. Teams should also prioritize solutions with built-in capabilities to reduce the variance in their experiment data using advanced methodologies like CUPED and to solve the multiple comparisons problem like the Bonferroni Correction.

Selecting the vendors with the most capabilities available today may not always be the best solution for your team. Highly specialized point solutions can create major headaches for under-resourced engineering teams when they have to manage duplicative data sets, build and

maintain expensive data pipelines, and manage kludgy integrations to run tests. It’s critical to understand the underlying technical challenges that may arise from adding another point solution to your stack.

Teams should define what they need to be successful, what resources they have available, and the maintenance requirements for the solution (or combination of solutions) they choose.

KEY QUESTIONS

- What statistical methodologies are used in the solution? Some examples include Sequential Testing, Fixed Horizon T-Test, Two-sided T-Test, and Bayesian Inference.
- Do I have the flexibility to alternate between different methodologies to design optimal experiments for my use case?
- If I peek at results during the test, can this skew my results?
- What types of experiments do I need to run to maximize my product investments?
- Will I need to build and maintain data pipelines or integrations to run effective experiments?
- If I do need to leverage data pipelines and integrations, which team(s) will be responsible for ongoing maintenance?

Managing user identity across devices for accurate targeting and allocation

One of the most crucial functional requirements for experimentation is the ability to manage user identity across devices and platforms - even if they are not logged in. Today's user has multiple devices and is often not logged in, which adds considerable complexity for teams trying to gain a clear understanding throughout each step of the customer journey.

While many solutions claim they can manage user identity, this requires more complex data mapping, deduplication, and scrubbing than product and engineering teams might recognize. Attempting to map devices across platforms and user identities quickly becomes extremely complicated at scale and using a disparate tech stack. All of these tools and data sets have to be stitched together using brittle integrations and expensive data pipelines. This leaves engineering teams in a difficult spot to maintain it and leaves analysts to try to make sense of it all.

Without the ability to harmonize user identity, it becomes next to impossible to confidently target experiments and releases across devices and platforms. When teams can't resolve user identity, it becomes far more likely to suffer degraded experiment results due to sample ratio mismatch or variant jumping.

But with natively integrated capabilities across analytics and experimentation, these challenges disappear since you have one unified data set. You have the flexibility to accurately target specific

users, cohorts of users, accounts, and even attributes like user operating system, device type, geography, user behavior, and more.

This also ensures teams can test using sticky bucketing meaning that users will only see one variant during a test regardless of which device they use.

The ability to resolve user identity is a critical piece to get right for every experimentation program. When product and engineering teams evaluate experimentation capabilities, this is one of the most important ones to get right. Otherwise, teams will struggle to successfully drive ROI from product-led experimentation.



Deliver reliable experiment data with automated data checks

One of the benefits of using pre-packaged experimentation platforms is the ability to improve data quality and reliability using automation. While teams can run stats packages on data sets in Python or R to determine statistical significance, this leaves teams open to missing potential data quality issues like sample ratio mismatch (SRM) or variant jumping. This is easily missed because using stats packages only evaluated the results of the test—not if the test was implemented correctly.

With built-in data guardrails, an ideal solution should automatically identify and notify teams about potential issues before they degrade experiment data—not after. By automatically checking for these issues in-app, teams can be confident that their results and approach which leads to trustworthy data and analysis.

KEY QUESTIONS

- Does the solution help teams remediate issues like sample ratio mismatch (SRM) or variant jumping?
- Does the solution support changepoint detection?
- Does the solution help identify potential outliers?

Best-in-class analytics and instrumentation for accurate and trusted results

You need robust data visualizations and deep analytics capabilities for successful experimentation. Analytics and tracking enable teams to interpret the results of an experiment and deeply understand the performance of each test drove. Without these capabilities, your team will have a difficult time deciphering what happened and will slow down your velocity since teams will spend more time debating what they should do next.

Some teams rely on an analytics point solution to visualize results and run deeper analyses. While this can work, it often results in teams getting double-charged for data events across multiple tools, wasting resources and budget. If teams can visualize data, drill down into key audiences, and get a deeper understanding of their experiments all in one platform, they are better positioned to make data-driven decisions faster and with less overhead.

Teams also benefit from the ability to seamlessly create and share dashboards about each experiment. The best solutions can summarize all of the relevant information about each experiment including the research behind the test, the initial hypothesis, targeting, analysis of each variant, and the ultimate decision made at the end of the test. This enables teams to easily share insights and improve collaboration. Easily shareable insights also help teams democratize learning across the entire product organization.

Teams should also consider how they will ensure standard metrics definitions across their stack, which often leads to inconsistent data. While this may seem like a trivial inconvenience, it quickly leads to painful consequences. Even if someone adds a filter to a data set in one platform, this can lead to inconsistent test results and harm data-driven cultures.

When executives and other teams see inconsistent data, no one can trust any shared insights, ultimately slowing teams down. While teams can try to alleviate this concern with data pipelines, the maintenance required to stand them up leads to significant headaches, high costs, and does not eliminate the risk of inconsistent data.

KEY QUESTIONS

- How do you visualize experiment results?
- How do I integrate your solution with my analytics platform?
- Will this integration lead to extra data consumption charges?
- How can I ensure consistent metrics definitions within my experiment across platforms?



Manage scaled programs and unblock teams with seamless program management and in-depth support

Teams looking to scale experimentation need to be able to manage their entire experimentation program, which could consist of dozens of experiments and releases simultaneously.

Each release goes through a specific set of milestones throughout its lifecycle and program managers need to be able to quickly understand what to do next rather than try to keep track of their programs on an ad hoc basis. Solutions that support sorting, filtering, and organizing experiments and releases help teams manage multiple experiments simultaneously.

Access to better education resources, professional services, user communities, and clear documentation unblocks and upskills team members more quickly than relying on a single person or team. These capabilities help experimentation programs scale.



Another consideration is supporting less experienced team members within the experience. Access to better education resources, professional services, user communities, and clear documentation unblocks and upskills team members more quickly than relying on a single person or team. These capabilities help experimentation programs scale.

KEY QUESTIONS

- How do I manage multiple experiments and releases simultaneously?
- Can I easily create, archive, or edit my experiments and releases?
- Where can team members get support if they get stuck?
- Are there support resources available like in-app guides, online learning modules, community forums, and documentation?

Safely deliver releases without sacrificing performance

Engineering teams need solutions that align to their preferred workflow and CI/CD processes. Best-in-class experimentation platforms leverage enterprise-grade feature management capabilities to deliver targeted experiences and releases safely. You also need the ability to process data in near real-time to truly understand each test's impact for on-the-fly analyses.

Another consideration is the latency that your tests will add to your user experience. Since latency will always negatively impact your conversion rates, this is a tradeoff teams need to understand as they build their experimentation programs. There can be scenarios where adding milliseconds of latency is worth it if you can incorporate user context into your tests. Teams must define this for themselves but should consider performance before investing in a solution.

KEY QUESTIONS

- How does the solution deliver experiments?
- How quickly does the solution process data?
- How much latency is introduced with the experimentation platform?

Best-in-class developer tooling and extensibility into my core systems

Developers need the right technical tools to successfully implement experimentation platforms. Specifically, they need client-side and server-side SDKs to implement tests in their environment's programming languages.

They will also need robust APIs to deliver experiments and manage releases programmatically. Teams should also consider what integrations they need to connect experiment data to their broader data ecosystem and tech stack.

KEY QUESTIONS

- Does the solution have client-side SDKs in the languages I need?
- Does the solution have server-side SDKs in the languages I need?
- Does the solution offer an API to manage experiments and release programmatically?
- Does the solution have integrations to my core systems like data warehouses and analytics platforms?

Enterprise-grade security, compliance, governance, and scalability

If your solution does not adhere to best-in-class security compliance, governance, and scalability, you will add significant risk to your organization. While these certifications and capabilities are not central to the core user experience, they are critical to your program's success.

Ensure you understand which security certifications your solution complies with, how robust user access controls are, the ability to audit changes to experiments and releases, and how well their data infrastructure can dynamically scale. You should also understand how many tests you can run simultaneously with your platform of choice.

KEY QUESTIONS

- What security certifications does the solution have today?
- Does it provide SSO?
- What level of roles-based access controls does it include?
- Can the platform scale dynamically at data volume?
- Are there any limits to the number of experiments I can perform at any time?



Bringing it all together: Critical capabilities to deliver experimentation at scale

FUNCTIONAL REQUIREMENTS

Experiment Design

CAPABILITIES	DESCRIPTION	PRIORITY? (Must have, Should have, Nice to have)	CAPABILITY RATING (0 TO 3)
Hypothesis generation	Ability to define hypothesis in-app		3 - Available 0 - Not available
Metrics selection	Ability to leverage key analytics metrics for experiments		3 - Available 0 - Not available
Guardrail metrics	Ability to select guardrail metrics within an experiment		3 - Available 0 - Not available
Sample size calculation	Ability to understand how long an experiment will need to run in order to reach statistical significance		3 - Available 0 - Not available
Experiment templates	A standardized set of metrics and segments for every experiment or release		3 - Available 0 - Not available
Holdout Groups	Designate traffic to never see experiments for baselining purposes		3 - Available 0 - Not available
Mutual Exclusion Groups	Separate colliding tests to ensure experiment traffic is not shared		3 - Available 0 - Not available
Long-running experiments	Ability to analyze tests outside of their experiment duration window and view historical results		3 - Available 0 - Not available

FUNCTIONAL REQUIREMENTS

Experimentation Use Cases

CAPABILITIES	DESCRIPTION	PRIORITY? (Must have, Should have, Nice to have)	CAPABILITY RATING (0 TO 3)
<p>Offers statistical analysis out of the box</p>	<p>Statistical approaches include:</p> <ul style="list-style-type: none"> • Sequential testing • Fixed Horizon T-test • Two-sided T-test • Bayesian Inference • CUPED • Bonferroni Correction 		<p>3 - Multiple methods available 2 - One method available 0 - Stats analysis outside of the application</p>
<p>Supports the experimentation use cases we need today</p>	<p>Common use cases include:</p> <ul style="list-style-type: none"> • A/B Tests • Multivariate testing • Nested experiments • Do no harm tests • Multi-armed bandits • Quasi-experimentation 		<p>3 - Supports all use cases I need 2 - Supports all but 1 use case 1 - Supports some use cases 0 - Supports none of my use cases</p>



FUNCTIONAL REQUIREMENTS

Allocation and Targeting

CAPABILITIES	DESCRIPTION	PRIORITY? (Must have, Should have, Nice to have)	CAPABILITY RATING (0 TO 3)
Managing user identity for known and anonymous users	Ability to connect user ID to device ID across platforms at scale without development effort		3 - Available natively 1 - Support requires data pipelines, integrations 0 - Not available
Cross-platform testing	Ability to test across devices and platforms natively		3 - Available natively 1 - Support requires data pipelines, integrations 0 - Not available
Targeting	Ability to target a variety of use cases natively: <ul style="list-style-type: none"> • By user • By user cohort • By any user property (platform, device, geo) • By user behavior 		3 - Available natively 1 - Support requires data pipelines, integrations 0 - Not available
Randomization	The methodology used to randomize traffic which can also include stratified sampling		3 - Available 0 - Not available
Sticky bucketing	Ensure users will see the same variant even if your experiment's targeting criteria, percentage rollout, or rollout weights are changed		3 - Available natively 1 - Support requires data pipelines, integrations 0 - Not available
Account-level bucketing	Ability to bucket traffic by account level rather than only user-level		3 - Available 0 - Not available



FUNCTIONAL REQUIREMENTS

Metrics, Analytics, and Data Visualization

CAPABILITIES	DESCRIPTION	PRIORITY? (Must have, Should have, Nice to have)	CAPABILITY RATING (0 TO 3)
Robust data visualizations	Ability to visualize results across a variety of charts		3 - Available natively 1 - Support requires data pipelines, integrations 0 - Not available
Statistical visualizations	Ability to visualize statistical concepts like confidence intervals, statistical significance achieved, and more		3 - Available natively 1 - Support requires data pipelines, integrations 0 - Not available
Drill down into results and visualizations	Ability to drill into charts for deeper insights		3 - Available natively 0 - Not available
Dashboards and reporting	Ability to add experiments to dashboards for knowledge sharing and collaboration		3 - Available natively 0 - Not available
Ability to segment by user behavior, any user cohort, group of users, or by any user property	Native segmentation for deeper insight into sub-groups that are under or over-performing relative to the rest of your users <ul style="list-style-type: none"> • By user behavior • By any user cohort • By any user property (platform, device, geo) 		3 - Available natively 0 - Not available
Recommendations on the next best action	Guided interpretation of your results and clear recommendations for your next best action		3 - Available 0 - Not available



View experiment or flag impact to any metric	Understand how each test impacts key KPIs using multiple types of data visualizations		3 - Available 0 - Not available
Metrics definitions	Ability to define and manage metrics within the UX without managing data pipelines		3 - Available natively 0 - Not available
Consistent metrics definitions across apps	Native ability to ensure consistency across apps		3 - Available natively 0 - Not available

FUNCTIONAL REQUIREMENTS

Data Guardrails

CAPABILITIES	DESCRIPTION	PRIORITY? (Must have, Should have, Nice to have)	CAPABILITY RATING (0 TO 3)
Proactively identify potential data quality errors	Debug and notify teams about critical data quality errors <ul style="list-style-type: none"> • Sample ratio mismatch (SRM) • Variant jumping • Exposure events without assignment events • Suspiciously high uplift detection • Changepoint detection • Outlier detection 		3 - Available natively 1 - Support requires data pipelines, integrations 0 - Not available

FUNCTIONAL REQUIREMENTS

Program Management

CAPABILITIES	DESCRIPTION	PRIORITY? (Must have, Should have, Nice to have)	CAPABILITY RATING (0 TO 3)
Lifecycle management	Manage simultaneous tests to understand the next required action and the number of experiments and releases across my team. Ability to filter active, archived, or completed tests.		3 - Available 0 - Not available
Lifecycle updates	Ability to rollout, rollback, create, delete, or edit tests without developer intervention		3 - Available 0 - Not available
In-app notifications	In-app guidance to remediate key issues or when key testing milestones are met including reaching stat sig, experiment completion, and more		3 - Available 0 - Not available
Support available in-app	Assist teams new to experimentation directly from the application		3 - Available 0 - Not available
Professional services	Offerings focused on implementation, training, and onboarding		3 - Available 0 - Not available
Self-paced learning resources	Access to interactive learning modules		3 - Available 0 - Not available
Access to user communities	Fosters a user community		3 - Available 0 - Not available
Online help and documentation	Access to up-to-date documentation and help resources		3 - Available 0 - Not available



FUNCTIONAL REQUIREMENTS

Delivery and Performance

CAPABILITIES	DESCRIPTION	PRIORITY? (Must have, Should have, Nice to have)	CAPABILITY RATING (0 TO 3)
Feature flags	Deliver experiments using feature flags for full stack experimentation		3 - Available 0 - Not available
Data processing in near real-time	Ability to analyze data at near real-time		3 - Refresh data at least once per minute 1 - Refresh data at least hourly 0 - Slower than one hour
Evaluation performance	Expected latency during an experiment and understanding of what tradeoffs are made		
Flexible approach to evaluation scenarios	Ability to select remote or local evaluation		3 - Available 0 - Not available
Experiment on user context with high-performance	Deliver highly performant experiments with user context		3 - Available 0 - Not available
JSON payloads	Ability to inject JavaScript into each variant for on-the-fly adjustments to text, colors, images, and more		3 - Available 0 - Not available
QA checks	Ability to QA tests or flags before they are rolled out		3 - Available 0 - Not available
Progressive rollout	Define allocation percentage for rollout		3 - Available 0 - Not available
Rollback	Ability to roll back a test and restore to a previous version if necessary		3 - Available 0 - Not available



FUNCTIONAL REQUIREMENTS

SDKs, APIs, and Extensibility

CAPABILITIES	DESCRIPTION	PRIORITY? (Must have, Should have, Nice to have)	CAPABILITY RATING (0 TO 3)
Client-side SDKs available	Languages: <ul style="list-style-type: none"> • Android • iOS • Web • React Native • _____ 		3 - Key languages available 2 - Most languages available 1 - Some available 0 - Not available
Server-side SDKs available	Languages: <ul style="list-style-type: none"> • Node • JavaScript • Python • _____ 		3 - Key languages available 2 - Most languages available 1 - Some available 0 - Not available
API support to manage experiments and flags programmatically	Offers API access import and export of data		3 - Available 0 - Not available
Integrations with your core technology	<ul style="list-style-type: none"> • Snowflake • Slack • Braze • _____ 		3 - Key integrations available 2 - Most integrations available 1 - Some available 0 - Not available
Integration with product analytics	Integration with analytics platforms like Amplitude		3 - Available natively 1 - Support requires data pipelines, integrations 0 - Not available
Recommendations on the next best action	Guided interpretation of your results and clear recommendations for your next best action		3 - Available 0 - Not available



FUNCTIONAL REQUIREMENTS

Security, Compliance, Governance, and Scalability

CAPABILITIES	DESCRIPTION	PRIORITY? (Must have, Should have, Nice to have)	CAPABILITY RATING (0 TO 3)
Security certifications critical to my business	<ul style="list-style-type: none"> • SOC 2 Type 1 • SOC 2 Type 2 • ISO27001 • ISO27018 • GDPR • CCPA • HIPAA 		3 - Offers all certifications I need 2 - Offers all but one key certification 1 - Offers some certifications I need 0 - Not secure enough for my business
Single Sign-On (SSO)	Offers SSO		3 - Available 0 - Not available
Data encryption	Ensures data is fully encrypted		3 - Available 0 - Not available
Robust user permissions	Offers robust roles-based access controls across teams and projects		3 - Available 0 - Not available
Audit of changes and updates	Ability to quickly understand when changes were made to flags and experiments		3 - Available 0 - Not available
Elastic scalability	Ability to scale dynamically as traffic increases		3 - Available 0 - Not available
Experimentation at scale	No limits to the number of tests that can be run in parallel		3 - Available 0 - Not available



Build vs. Buy: A Guide to Selecting Your Experimentation Platform



By Chad Sanderson

For a growing company to thrive and remain competitive, it needs experimentation at the core of its product development process. Experimentation capabilities allow companies to run product performance tests, collect data, and analyze results. They provide insights about user behavior and preferences that help companies create sticky user experiences, improve conversions, and drive engagement.

A frequently asked question is: Should a company build its experimentation tool or purchase one from a third-party vendor? The age-old debate of build vs. buy is challenging due to the complexity of experimentation platforms, the costs and resources required, and how each option aligns with your overall strategy.

I have worked on data experimentation with companies like Convoy, Microsoft, and SEPHORA, where I have had to build and buy experimentation platforms. In this chapter, I will discuss the most common trade-offs product and engineering teams should consider when deciding whether to build or buy an experimentation platform.



Successful experimentation platforms enable faster innovation, data-driven cultures, and better stakeholder alignment

Before discussing the specific tradeoffs associated with each choice, let's start with what success looks like. Success is validating new ideas before fully deploying them using an experimentation platform. This is a game-changing and magical experience for many product and engineering teams.

Without an experimentation platform, product and engineering teams launch new features without knowing how their core metrics are impacted. There's a lot of guesswork involved. Experimentation tools allow teams to easily and concretely measure performance, simplifying their decision-making process on what to invest in.

Before teams decide whether to build vs. buy, they need to define the key metrics and KPIs they will measure. If your experimentation tool doesn't let you measure the impact of testing with key business metrics—like revenue or profit margin—it's hard to justify the ROI of experimentation in general.

We needed to overcome the measurement challenge while I was at Convoy. At Convoy, the entities we cared most about were shipments, contracts, facilities, and geographic regions. As we began to evaluate the build vs. buy question, we needed experimentation capabilities to align with our product development philosophy. That is, we needed an experiment for any feature deployment or release that could affect any of these entities. This was a mandatory requirement for our build vs. buy decision.

The right experimentation capabilities have advantages beyond faster innovation. They can also help win buy-in from leadership and other stakeholders.

The right experimentation capabilities have advantages beyond faster innovation. They can also help win buy-in from leadership and other stakeholders. For example, a CPO often needs to justify additional investment and resourcing from their CFO. Experimentation platforms can provide beneficial reporting and analytics to highlight which product updates have generated meaningful ROI and how new investments can continue delivering innovation and value to the organization.

How to approach the build vs. buy decision

When organizations recognize they need to invest in experimentation capabilities, they face a decision: Do I build a tool from scratch, or do I buy a pre-packaged solution instead?

Product and engineering teams should evaluate these five factors when they consider the build vs. buy question:

- 1. Cost:** How much will buying an out-of-the-box solution cost compared to building your in-house solution? When determining the cost of an in-house solution, make sure to account for line items like hiring and training developers, hardware and software expenses, integrations, and ongoing maintenance. For pre-packaged software, consider licensing costs as well as training and services.
- 2. Time:** How long will building your software in-house take as opposed to purchasing and onboarding a pre-built solution?

3. **Expertise:** Does your current team have the technical know-how to build and maintain experimentation software?
4. **Customization:** Do the pre-built solutions available meet your company's specific needs and requirements?
5. **Competitive advantage:** Consider whether building software in-house or a pre-built solution will offer features that help you build better products than your competitors.

Ultimately, the decision to buy or build software depends on your specific needs and the resources you have available. Careful evaluation of these factors can help determine the best option for your organization.

The “build” argument: Pros and cons

Building your experimentation solution might sound like a good idea, particularly if you have the resources to do it, but it has its trade-offs. Here are some pros and cons of building in-house based on the five factors mentioned above:

The advantages of building your experimentation platform include:

- ✓ **It allows for more customization.** If your company has unique needs that existing experimentation platforms cannot meet, it may be necessary to build your platform. For example, if you need to integrate with a specific data source or use a custom statistical algorithm, building your own platform may be the best option.

- ✓ **It gives you access to all of your data.** Building your solution gives you access to your own databases, data lakes, or data warehouses. You can carry out anything from aggregations to writing SQL—your data never has to leave your system.
- ✓ **It might give you a competitive advantage.** Building your platform gives you flexibility. You can customize the tool to meet the changing needs of your organization. In the process, you may have created software that addresses a need not yet solved in the market.
- ✓ **If you deal with large volumes of experiments and experiment data, it can be more cost-effective.** By building your platform, you'll have more control over resources and will be less dependent on external providers.

But there are some considerable drawbacks to building your solution:

- ✗ **It is very expensive.** Building your experimentation platform can be costly and time-consuming. If you're building your platform internally, that's a minimum of three to four engineers, one to two data scientists, a product person, and a manager to oversee the project. This can cost organizations millions of dollars to maintain that system every year.
- ✗ **It is time-consuming.** Building from the ground up will take significantly longer than buying a solution out of the box and could take several quarters to create a functioning MVP, let alone a production-ready solution.

❌ **It requires specific expertise.** Building your experimentation platform requires a high level of expertise in software development, data analysis, and statistical modeling. If you have a team with the necessary skills, building your own platform may be a viable option, but your team members will need to spend the bulk of their time on this project. The other challenge is having a backfill plan to replace this expertise if a team member leaves your organization.

Building your experimentation solution requires significant expertise and resources, but it may be worthwhile in the right organization.

The “buy” argument: Pros and cons

Alternatively, there are several reasons why it may be better to use an existing experimentation platform. Out-of-the-box solutions get the job done for the majority of companies since many company requirements are not that complicated.

Here are a few pros and cons of buying pre-built experimentation software based on the five factors mentioned earlier:

The advantages of buying an experimentation platform include:

✔ It is highly cost-effective. More often than not, using an existing experimentation platform is significantly cheaper than building one yourself.

- ✔ It requires much less time to get started. One of the major advantages of purchasing out-of-the-box solutions is that you can get started immediately, particularly if you find a solution with short onboarding timeframes.
- ✔ It’s not as time-consuming for your dev teams. Building your own solution will take up most, if not all, of your development team’s time and resources. Buying a pre-built solution frees them to focus on running experiments and other business-critical tasks.
- ✔ It doesn’t require specific expertise. The main reason you would want to build your own solution is that you have robust and complicated feature requirements and the resources available to execute. In these cases, it is unlikely that your sales or marketing teams will figure out how to use it. Ready-built solutions are easy to use, which helps organizations achieve data democratization.
- ✔ You still have access to experts. Most pre-built software plans provide access to experts who have already developed and tested the software.
- ✔ Many pre-built solutions are designed to have a degree of flexibility and customization. It might not be as much customization as an in-house solution, but you may still be able to achieve your desired functionality.
- ✔ Quicker time to market. The time you save in purchasing a pre-built solution means you enjoy much faster time to market, which will give you an edge over competitors.

While there are considerable advantages to buying an experimentation platform, a few tradeoffs also exist:

- ✘ There are typically fewer customization options. But it's important to consider whether customization is necessary and whether you can achieve it on some level with a pre-built solution.
- ✘ No competitive advantage. You may not have the competitive edge of a unique software, but this might not be a priority, especially if a pre-built solution provides more functionality than what you could develop in-house.

Established experimentation platforms have also been tested and used by a large number of users, so they're often more reliable and stable than a custom-built solution. And most out-of-the-box solutions come with easier integration requirements with other in-house solutions you're running.

Making the decision: Build or buy?

Ultimately, whether or not to build your experimentation platform from scratch or buy an out-of-the-box solution comes down to two factors—purpose and resources. You first need to figure out what your requirements are. If something on the market meets those requirements, you have your answer.

It simply doesn't make much sense to build something that another product is already doing better and will cost you less. Pre-built solutions are the product of massive teams of experts with diverse skill sets who do all the legwork, so you don't have to. Since these products are built at scale, they are also more cost-effective.

Considering the huge costs of building your experimentation solution, organizations should only opt for this route when the opportunity cost of not building exactly what you need is very high. But if you evaluate this critical decision based on the five key factors—cost, time, in-house expertise, customization needs, and competitive advantage—your organization can create a data-driven product culture.



How Amplitude Experiment Guides You to Scale Experimentation



By Audrey Xu
Amplitude

In this chapter, we will show how product, data, and engineering teams can work together to scale product-led experimentation. You will learn how Amplitude's unified Digital Analytics Platform drives efficient innovation through natively integrated experimentation with self-serve analytics.

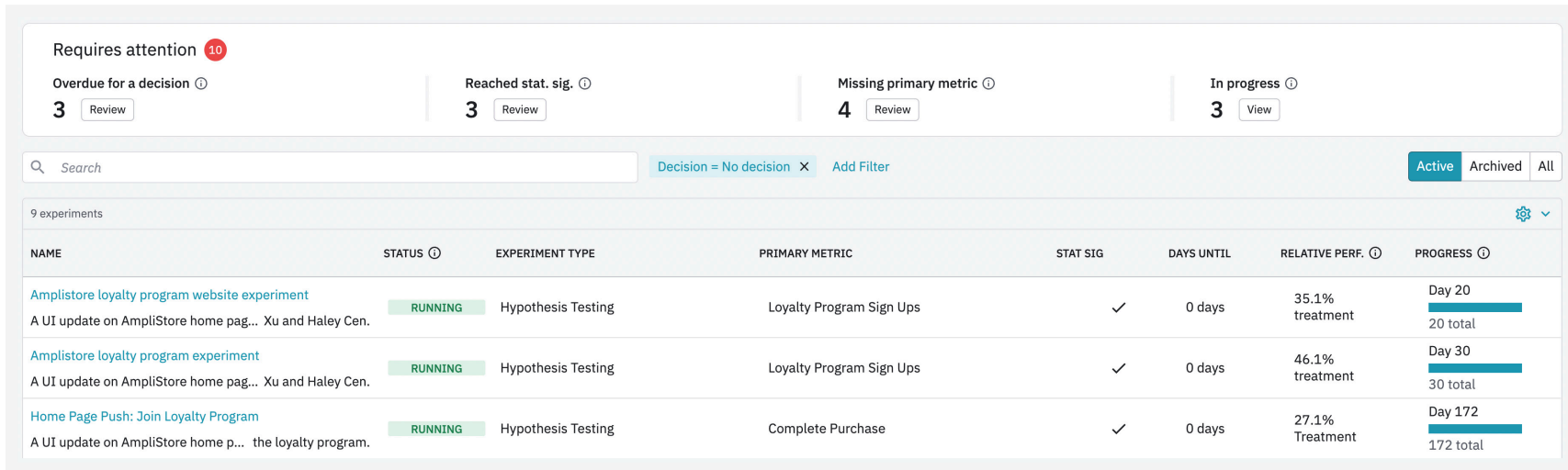
Today, we are part of the product organization at AmpliStore. At AmpliStore, our focus is to encourage more users to join our loyalty program. This is a key goal for our business since Amplitude Analytics has demonstrated that users who join the loyalty program have higher revenue per user and better retention.

For our demonstration, we will highlight how Product Managers, Data Analysts, and Engineers can work together to build a culture of experimentation built into the core of their product development process.

Amplitude's unified Digital Analytics Platform drives efficient innovation through natively integrated experimentation with self-serve analytics.



Maximize your experimentation program with end-to-end lifecycle management



Product teams understand the value experimentation delivers to their product experience. But managing a scaling experimentation program is challenging when your team could be running dozens of experiments at a time. Product teams need to understand the status of every test running and be guided to their next best action in one view. This is exactly what [Amplitude Experiment](#) was designed to do.

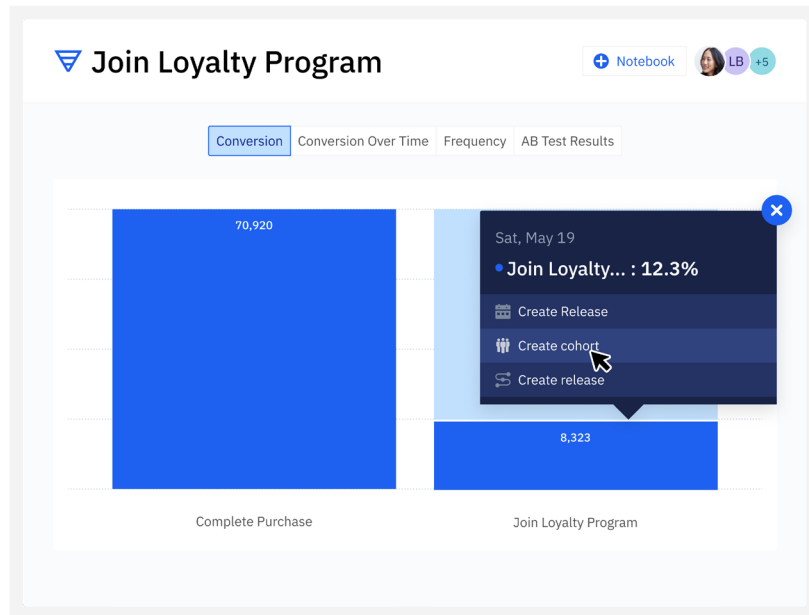
When you open the Experiment homepage, you will see an at-a-glance overview of every experiment in your program. At the top of the page, the Summary Card provides guidance about which experiments are ready for a decision, making it easy to manage Amplitude's entire experimentation program from the homepage.

After reviewing the loyalty experiment, it's clear this experiment variant won based on its performance relative to our control variant. Now, we can easily roll out this variant to the rest of our users in a phased roll out. We can also add this to our [Notebook](#), which provides the rest of our product organization a clear view into the insights that guided us to run this experiment, the hypothesis we created, the metrics used, our variants, and ultimately the outcome of the test.

Notebooks help us build a culture of experimentation and learning at Amplitude by democratizing our learnings to the rest of the team, improving collaboration.

Ensure high-quality experiments and trusted results with a workflow-based UX

As a Product Manager at AmpliStore, I am constantly using Amplitude Analytics to better understand what users are doing (or not doing) in my product. I create a [Funnel Analysis](#) chart to understand how often users join the loyalty program after completing a purchase at AmpliStore.



I quickly realize that this cohort of users is not joining our loyalty program. Conversion rates are currently at 12.3%, but we had expected closer to 24% conversion based on the exclusive discounts

we offered as part of a recent release. These users are still completing purchases but not joining the loyalty program—the conversion event we want to drive.

I use the Microscope and create a new cohort—“Users who completed purchase but did not join loyalty program”—with one click. This cohort is now available for me to run a targeted experiment on them since these capabilities are natively integrated in Amplitude’s Digital Analytics Platform.

I take these insights to our engineering and design teams to have them build out the code for our test. While they build the code, I select “Create new Experiment” in Amplitude. When I create a new experiment, I start in the Plan tab. Amplitude Experiment incorporates a workflow-based design built to ensure teams adhere to best practices for experimentation. One of the primary ways teams suffer from degraded results is from poor experiment design. Our product is purposefully designed to automatically incorporate these best practices through our user experience, ensuring teams can trust their results at every step.

Our product is purposefully designed to automatically incorporate these best practices through our user experience, ensuring teams can trust their results at every step.

Next, we add our primary metric, “Loyalty Program Signups” to our experiment. The drop-down shows me that every event and metric tracked in Amplitude Analytics is automatically available to us in Experiment.

The screenshot shows the 'Experiment Goals' configuration page. It includes sections for 'Experiment Type' (Hypothesis Testing selected), 'Bucketing Unit' (User selected), 'Exposure Event' (Amplitude Exposure selected), and 'Metrics' (Loyalty Program Sign Ups selected with an increase direction and a 15% goal).

That means we do not need to worry about consistent metrics definitions across our analytics and experimentation platforms—a common and potentially costly challenge when teams have disparate tech stacks. With a unified stack, teams no longer have to build and manage data pipelines or suffer from targeting limitations.

I now have access to our built-in Sample Size Calculator to understand how long this test will need to run before we reach statistical significance based on AmpliStore’s traffic. We select “Install App” as our proxy metric since we assume that this action is correlated with “Loyalty Program Sign Up” since customers downloading our app are more likely to join our loyalty program than customers who do not.

The screenshot shows the 'Sample Size Calculator' interface. It displays an estimated sample size of 8,539 users to reach significance. The primary metric is 'Loyalty Program Sign Ups' and the proxy exposure event is 'Install App'. The minimum effect (MDE) is set to 15% and the confidence level is 95%.

Deliver targeted experiments with full control

After engineering finishes building and reviewing the code, they configure the feature flag to the appropriate deployment and add the code for each variant directly into the user interface. Amplitude Experiment delivers each test using flags, ensuring each variant is delivered safely to each end user.

Allocation [Allocation Guide](#)

Who should be eligible for this experiment?

All Users Targeted Users

Segments are evaluated in order top-to-bottom. If a user doesn't match the first segment, then subsequent segments (in order) will be evaluated. If a user does not match any segment, they will not receive any variant and should see your default product experience; these users will not be recorded for statistical purposes.

Allow rollout controls per segment ⓘ

1 Segment 1 Filter × :

where Cohort = Users who completed purchase but did not join loyalty program ⓘ ×

[+ Add Segment](#)

What percentage of the targeted segments should be included in this experiment?

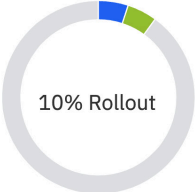
%

Bucketed by ID

How should traffic be distributed between the variants?

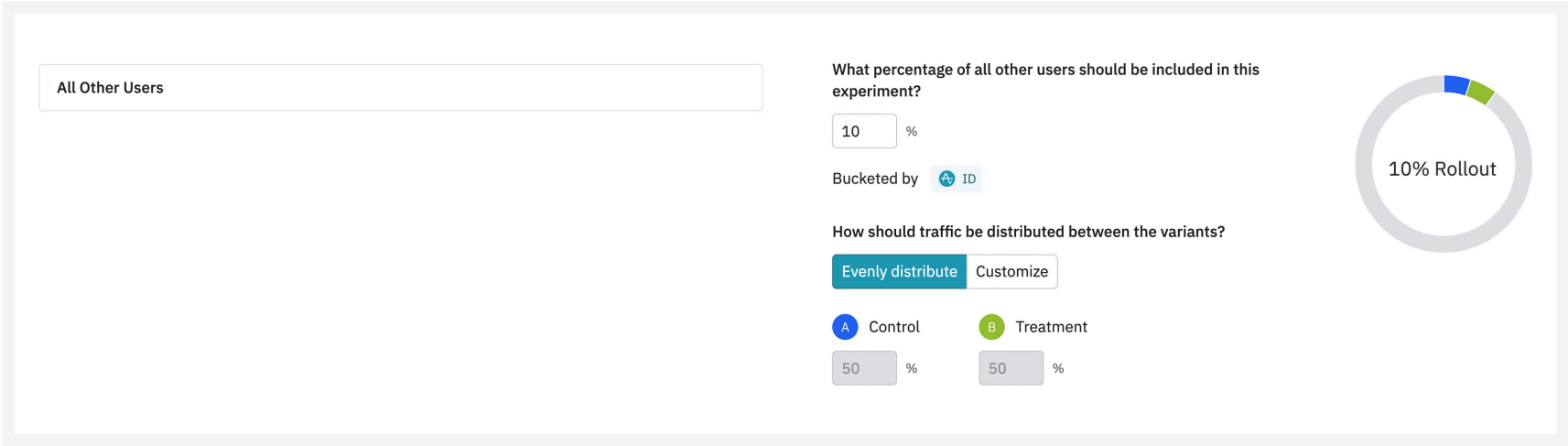
A Control B Treatment

% %



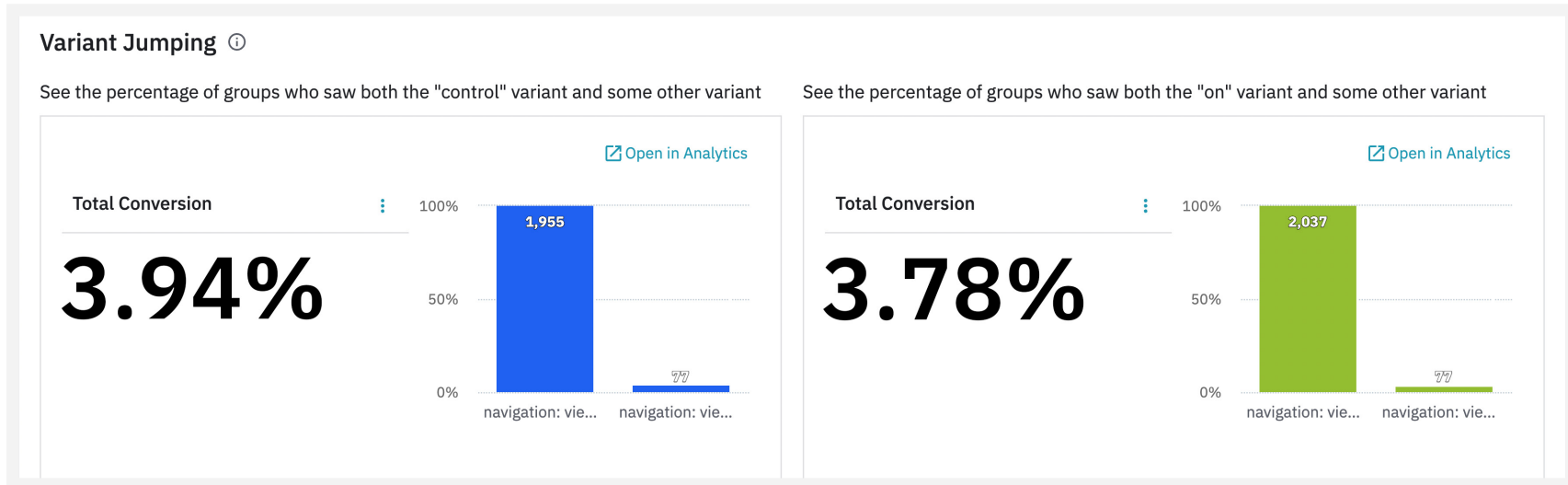
I configure the roll-out percentages and add the “Users who completed purchase but did not join loyalty program” cohort identified in Analytics for targeting. This is one of three ways teams can target users within Amplitude—user device ID, rule-based segments, or non-targeted users.

Now that our experiment is set up, we can activate the experiment.



A few days later, I want to see how the experiment is going. Amplitude Experiment uses [sequential testing for our stats analysis](#), so I can safely “peek” at the results as they happen without worrying about skewing our experiment data. This is a major advantage for product teams since it allows them to quickly understand the progress of the test as it happens rather than waiting until the experiment is completed like with a fixed-horizon T-Test.

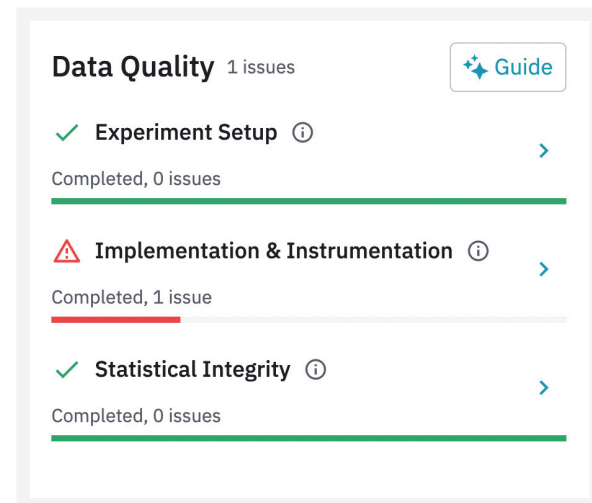
Remediate data quality issues faster to improve your results



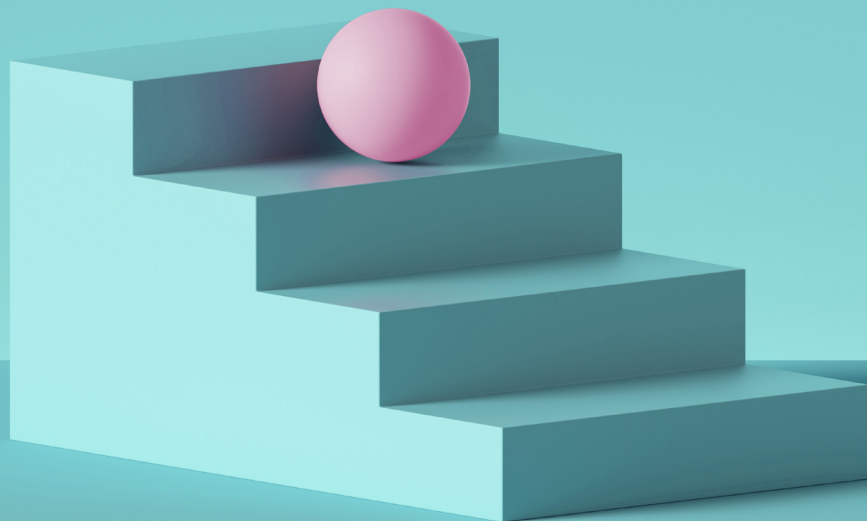
A week into the test, I get an email notifying me of a potential issue with the Implementation & Instrumentation of our experiment. Our data quality checks are highlighting that our test may be experiencing Variant Jumping, a major we might be suffering from Variant Jumping, a major problem with our results.

I head to the Monitor tab and immediately notice an issue: exposures and assignments are supposed to be at 50/50 but are misaligned. This means some users likely see both variants within the same test. Since experimentation requires completely randomized and independent data sets, we probably have some serious issues with our results.

The Analyze tab shows that the data quality checklist clearly highlights the issue, guiding me to quickly remediate it.



Since Analytics and Experiment are natively integrated, it is easy to identify challenges, get notified to act, and resolve the issue with help from engineering.



I work with engineering to resolve the issue quickly. Engineering seamlessly rolls back the test and restores the previous version using Experiment's robust feature flagging capabilities. Engineering opens up Analytics and realizes an implementation issue with Android users. Amplitude tracks user "platform" as a user property and seamlessly manages user identity across devices, making it easy for engineering to identify which user cohort experienced the issue.

Engineering rewrites the code to fix the issue. Since Analytics and Experiment are natively integrated, it is easy to identify challenges, get notified to act, and resolve the issue with help from engineering.

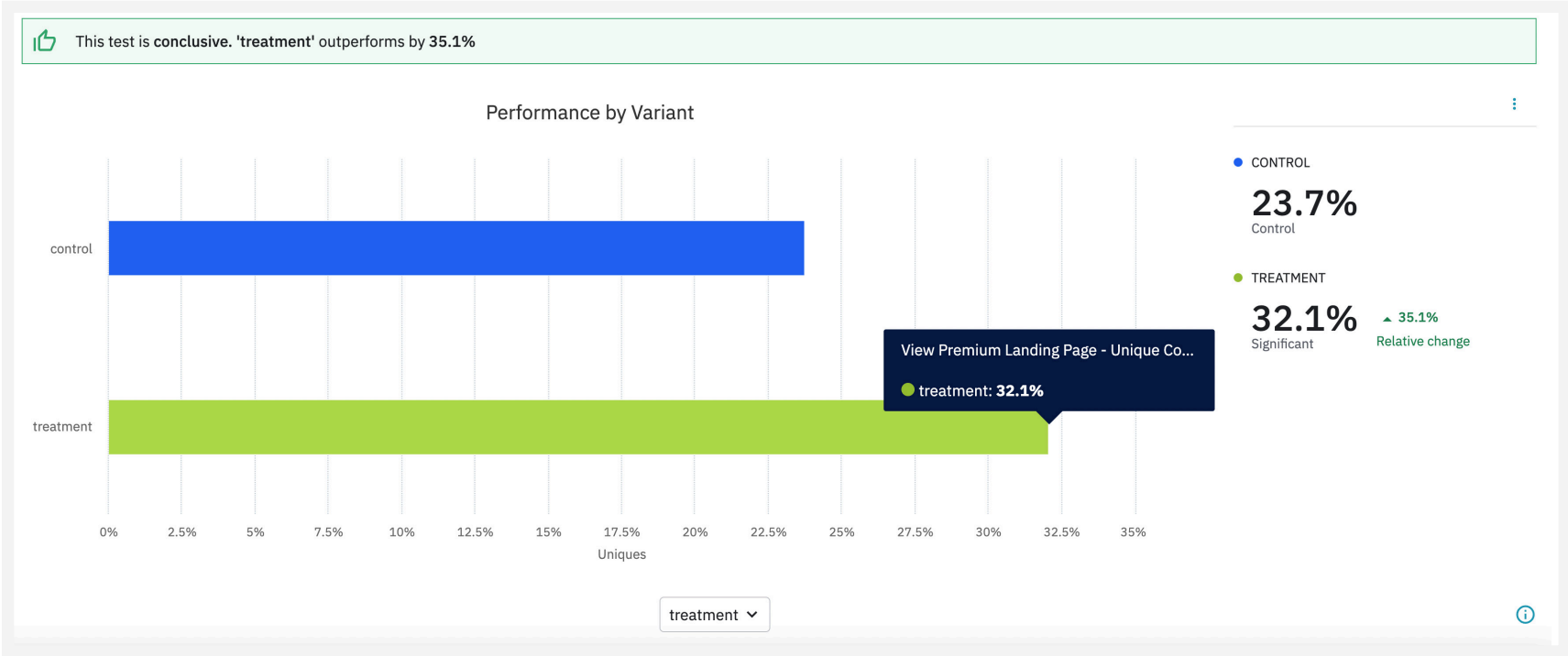
And now, I am ready to re-run the experiment.

Analyze your results and safely deliver the new experience to your users

After re-running our test and letting it run for two weeks, I am automatically notified by email that we have achieved statistical significance. I connect with my data analyst team to review the results.

Since our analysts understand and trust Sequential Testing and the data quality checks built into Amplitude Experiment, they are comfortable with the test results. This saves our data analysts a significant amount of time and allows them to focus on higher-value analyses while unblocking me to innovate our product experience faster.

Based on our test, we achieved a statistically significant lift in our loyalty program sign up conversion from 23.7% to over 32.1% based on our changes—and much closer to our business targets.



After I meet with my data and engineering counterparts, we are ready to roll out the winner to more users. Engineering uses a progressive roll out to increase the traffic from 10% to 100% of users, which gives us the confidence that our efforts will Loyalty Program signups and add significant value to our business.

AmpliStore's product organization can now scale our experimentation program by connecting analytics and experimentation in one unified platform. This helps us accelerate innovation and maximize our product investments. We can also significantly reduce product development cycles through better collaboration across product management, engineering, and data teams.

Complete Experiment ✕

Decide how to take action on your experiment

Decision *

Pick Variant *

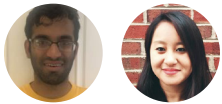
Rollout To *

All Users
 Custom

All segments, including untargeted users, will be set to a 100% rollout of the selected variant. Sticky bucketing is turned off.

Reason

How Experimentation Works in Amplitude



By Akhil Prakash and Kathy Qian
Amplitude

Built with product-led experimentation in mind, [Amplitude Experiment](#) brings the same rigor to experimentation that distinguishes [Amplitude Analytics](#). In this chapter, we'll explore how the platform works, including its testing methodology and the automated features and other tools designed to help you conduct scientifically sound tests that yield results you can trust.

Testing methodology

Most experiments fall into two categories: “hypothesis testing” and “do no harm.” “Hypothesis testing” refers to experiments that use data to determine which variant to roll out based on performance. If none of your variants outperform the control, then in most cases, it makes sense to stick with the control experience.

“Do no harm” experiments are used to confirm that a change will not significantly harm key metrics. This type of experiment is often used for design system alterations or to mitigate risk.



Of course, there are cases where an experiment does not reach statistical significance. When this happens in hypothesis testing, Amplitude recommends reverting to the control. For do no harm experiments, we recommend rolling out the higher-performing variant based on the direction of the primary metric. If the direction is “increase,” roll out the treatment with the most positive lift; if the direction is “decrease,” roll out the variant with the most negative lift.

Sequential testing

Amplitude Experiment offers multiple statistical methodologies for hypothesis testing that can be configured directly from the user experience. One of the most popular methodologies used by product

teams is [sequential testing](#) due to its built-in advantages, chiefly the ability to “peek” at the results throughout the experiment, without degrading your results. Our specific version of sequential testing, [mixture Sequential Probability Ratio Test \(mSPRT\)](#), allows you to peek as many times as you want—and you don’t have to decide on a number before the test starts, as you would with a grouped sequential test. As a result, you can decrease the experiment duration if the effect size is much bigger than the minimum detectable effect (MDE).

Another advantage of sequential testing is that it does not require users to know how to use a sample-size calculator, which can be challenging for non-technical team members. Instead, Amplitude pre-populates the sample-size calculator with standard industry defaults (95% confidence level and 80% power), computing the control mean and standard deviation (if necessary) over the last seven days.

In sample-size calculators, there is typically a field called “power” (1- false negative rate). With sequential testing, this field is essentially replaced with “how many days you are willing to run the test for.” This is a much more accessible number for most product and engineering teams.

T-tests

In addition to sequential testing, Amplitude Experiment offers [fixed horizon T-testing](#), a standard methodology familiar to any data scientist. T-tests are valuable for their precision and are best used when sample sizes are small. They can also be useful when ending an experiment

early might risk skewing the results, such as overweighting certain days or weeks—an issue where a product’s seasonality comes into play. For instance, a map app may see higher traffic on weekends as opposed to weekdays. T-tests are also the go-to for any experiments where the object is to study long-term metrics.

Bonferroni correction

Simple, single-hypothesis tests can yield valuable insights, but multiple-hypothesis tests are often even more useful and certainly more efficient. There is one downside: they can introduce errors into your statistical significance calculations via the [multiple comparisons problem](#) (also known as multiplicity or the look-elsewhere effect). The probability of making an error (by basing a critical business decision on a false positive result) increases rapidly with the number of hypothesis tests you are running.

Fortunately, there are statistical tools used to compensate and correct for the multiple comparisons problem. Amplitude uses the Bonferroni correction to accomplish this.

The Bonferroni correction is the simplest statistical method for counteracting the multiple comparisons problem. (It’s also one of the more conservative methods and carries a greater risk of false negatives than other techniques.) Mathematically, the Bonferroni correction works by dividing the false positive rate by the number of hypothesis tests you are running; this is equivalent to multiplying the p-value by the number of hypothesis tests.

Amplitude Experiment performs Bonferroni corrections on both the number of treatments and the number of metrics in each of the two metric tiers (primary and secondary). In other words, Bonferroni is only applied to the primary metric when there are multiple treatments (i.e., more than two). Bonferroni is applied to the secondary metric if there are multiple secondary metrics or multiple treatments.

Look for an info icon in the significance column when Bonferroni correction is applied. The tooltip shows the corrected and uncorrected p-value.

CUPED

In traditional [A/B testing](#), the average treatment effect is estimated by comparing the average outcomes of a treatment group to a control group. However, this method assumes that the treatment effect is the same for all individuals, which is not always true.

To address this limitation, Amplitude Experiment uses the statistical technique [Controlled-experiment Using Pre-Existing Data](#) (also known as [CUPED](#)), which estimates the treatment effect separately for each individual and then aggregates the individual estimates to obtain an overall estimate of the treatment effect. This ensures CUPED helps you reduce variance in your tests and achieve statistical significance faster.

How? CUPED first identifies a baseline characteristic (also known as a covariate) that may be related to the treatment effect. This covariate is then used to match individuals in the treatment and control groups based on their propensity score, which is the predicted probability of performing the conversion event based on the covariate.

By matching individuals with similar propensity scores, CUPED corrects for differences in users' likelihood to convert that stem from factors other than the product change being tested, which reduces the bias in the estimated treatment effect. This is important because some subgroups are more likely to respond to the treatment than others, and these subgroups may not be equally distributed among treatment and control groups.

There are some cases where CUPED is not necessary or will not reduce variance within your tests. This is true if you are only targeting new users in your test or the event was not instrumented in Amplitude Analytics during the pre-period. In general, anonymous users can be problematic for CUPED, but with Amplitude's differentiated approach to [seamlessly managing user identity](#), this is not a problem for Amplitude Experiment customers.

Toggle on CUPED within your statistical settings under the Analyze tab in Amplitude Experiment. This is also available within Experiment Results.

Targeting and assignment

Ensuring the right users are exposed to the right variant is essential to the integrity of your experiments—and your results. That’s why we’ve built Amplitude Experiment with a variety of tools and capabilities designed to do just that.

Exposure

One of the most important concepts to understand is [exposure events](#). An exposure event is a strictly defined analytics event sent to Amplitude to inform Amplitude Experiment that a user was shown a variant of an [experiment or feature flag](#). Exposure events contain the flag key and the variant of the flag or experiment that the user has been exposed to in the event properties.

When Amplitude ingests an exposure event, it uses the flag key and variant to set or unset user properties on the user associated with the event. Setting user properties is essential for experiment analysis queries on primary and secondary success metrics.

Sticky bucketing

Teams also need to ensure that users have a consistent experience regardless of what devices they use. [Sticky bucketing](#) helps ensure that a user will continue to see the same variant if your experiment’s targeting criteria, percentage rollout, or rollout weights are changed. Amplitude Experiment uses [consistent bucketing](#), which keeps users bucketed into their original variants as long as you don’t change anything. (Note: Amplitude Experiment uses a [deterministic hashing algorithm](#), not a random hashing algorithm.)



When sticky bucketing is on, Amplitude Experiment will not evaluate users based on targeting conditions or allocation percent. Instead, they will continue to see the last variant they saw. (See the [evaluation flow](#) chart to learn more about the order in which evaluation happens.)

Sticky bucketing is often used as a defense mechanism against [variant jumping](#). However, simply enabling sticky bucketing does not guarantee you’ll never see variant jumping. It may still occur if your experiment includes both a logged-out and a logged-in experience. When the user is logged out, they may have a different Amplitude ID than when they are logged in.

To turn sticky bucketing on or off, navigate to the Configure tab and look for the sticky bucketing toggle under Advanced Settings.

When should you enable sticky bucketing?

- You want to give the user a consistent experience, even if the user property you're targeting changes. For example, if you're running an experiment only in the United States, enabling sticky bucketing would ensure your users would see the same variant if they happened to travel outside the country.
- You want to decrease the percentage of users in an experiment where the treatment group is not performing well. But you don't want users to be moved from either the treatment or the control to the group that never saw either variant. Enabling sticky bucketing will keep users in their assigned groups even after you change the percentage rollout.
- You want to target users for a specified duration and then stop targeting new users while maintaining the original assignments for any users that have already been bucketed. Enable sticky bucketing at the beginning of the experiment with a 50/50 split. Then, after the duration passes, change the rollout percentage to zero.
- You want to sunset a failed experiment but ensure the users bucketed into an experience still get that experience. Enable sticky bucketing and set the rollout percentages to zero.

Sticky bucketing helps ensure that a user will continue to see the same variant if your experiment's targeting criteria, percentage rollout, or rollout weights are changed.

Do not enable sticky bucketing when:

- You want the user's experience to change as the targeted user property changes. To continue an example from the previous list, if you're running an experiment in the United States, you may not want users to have the same experience if they're traveling abroad. There may be legal reasons you cannot enable certain features of your app in certain countries, or there may be localization issues that affect how your app's UI displays.
- Your experiment is intended to drive free users to become paid users and relies on earning rewards. Once these users convert, you no longer need to offer a reward. If sticky bucketing were enabled here, those users would receive the free experience even after upgrading to paid.
- You want to enforce a "cool down" period between giving discounts. If you want to limit the frequency of discounts for each user to once every seven days, you can add a seven-day filter to the targeting criteria; if a user received a discount within that period, the flag would evaluate to [off]. With sticky bucketing enabled, this would not happen, and the user would collect another discount before you wanted them to.
- You are rolling out or rolling back a variant. When sticky bucketing is enabled and you change the traffic allocation, you'll get a weighted average between the old and new allocation (since the users who were previously bucketed will stay in their bucket). It will take some time for your experiment to achieve the desired allocation.

This is not intended to be an exhaustive list. There are also cases where the results would be the same, regardless of whether sticky bucketing was on or off. An example might be an experiment where you're targeting everyone who views your home page, and you do not touch any of the experiment controls while the experiment is running.

Bucket on group IDs

Experiment Analysis supports different units of analysis. Previously, the default unit was "user." Now we also support various group types like org ID, account ID, etc. This level of flexibility is beneficial for B2B organizations who want to deliver tests to specific accounts, such as beta customers, or perhaps restrict high-value customers from receiving any experiments at all.

Holdout groups

Teams need a variety of ways to manage running multiple tests at a time. Using holdout groups is one way to achieve this. To measure the long-term and combined impact of multiple experiments, it is useful to withhold a portion of users from any experiment. This approach yields a more comprehensive understanding of the effects, as statistical significance in a single experiment may not reflect the overall impact. Amplitude Experiment makes it easy to exclude a group of users from your experiments by creating a holdout group. This will withhold a portion of the overall traffic from seeing any experiment within the group.

We recommend adding experiments to a holdout group for the following use cases:

- Measuring the long-term impact of your rolled out variants
- Measuring the lift of your team's product changes as a whole

Best practices

The following are some best practices to keep in mind when using holdout groups:

- Set the holdout percentage to a value between 1-10%.
 - It is recommended to have a large amount of traffic to begin with, otherwise withholding a significant portion of your total traffic will lead to extended experiment time frames.
- Don't add a running experiment to a holdout group. This may severely compromise the integrity of your data because it may unassign users from the active experiments being added.
 - We recommend adding experiments to a holdout group before they have started running.
- Don't remove a running experiment from a holdout group. This may compromise the integrity of your data because it may assign users to the active experiments being removed.
 - Deleting a holdout group with running experiments has the same consequences. Delete the holdout group after all experiments in the group have concluded.

Mutual exclusion groups help ensure users included in one experiment are not exposed to any related experiments (or colliding tests) at the same time.

Mutual exclusion groups

Another way for teams to manage running multiple tests at a time is to use [Mutual Exclusion Groups](#). Mutual exclusion groups help ensure users included in one experiment are not exposed to any related experiments (or colliding tests) at the same time. This is crucial to avoid interaction effects or conflicting results that may arise from running multiple experiments simultaneously to solve the same problem.

With Amplitude Experiment, you can set two or more experiments to be mutually exclusive. Simply add both experiments to the same exclusion group. Amplitude Experiment will take care of the rest.

We recommend mutually exclusive experiments for the following situations:

- When simultaneous experiments occur in the same area of your product and have the same goal.
- When simultaneous experiments occur in the same funnel and have the same goal.

Alternatively, you could run these experiments one after the other instead of simultaneously.

Best practices

The following are some best practices to keep in mind when using mutual exclusion groups:

- Evenly distribute traffic between your slots.
- Don't add a running experiment to a mutual exclusion group. This may severely compromise the integrity of your data because it may unassign users from the active experiments being added.
 - We recommend adding experiments to a mutual exclusion group before they have started running.
- Don't remove a running experiment from a mutual exclusion group. This may compromise the integrity of your data because it exposes your users to the other experiments in the group.
 - In addition, deleting a mutual exclusion group with running experiments has the same consequences. We recommend deleting the mutual exclusion group after all experiments in the group have concluded.

Sample Size Calculator

Calculating the right sample size—the number of participants needed for an experiment—is another important step toward accurate results.

Amplitude's [sample size calculator](#) helps you determine how much traffic each variant needs to reach statistical significance for a given metric. Once you have this number, you can divide it by the average traffic per day or week for the segments you plan on targeting to estimate how long the experiment will likely take. (While Amplitude

Experiment supports sequential testing, the sample size calculator solely supports determining the sample size for T-tests.)

If you find that the sample size returned by the calculator is larger than you'd like, resulting in an experiment that would take too long to execute, there are a few steps you can take:

Increase the minimum detectable effect (or MDE). The lower the MDE, the more difficult it is to measure precisely, meaning you'll need a larger sample size. To understand what you should use as a MDE, it's helpful to think about the ROI of an initiative and what magnitude of increase in your primary metric is needed to justify the cost of implementing and maintaining the change.

Decrease the confidence level. Generally, 95% is accepted as the industry standard. However, values as low as 80% can help generate directional insights.

Experiment duration estimation

The experiment duration estimate is designed to predict the length of time your experiment will run. It can only be used with the primary metric but works for both sequential testing and T-tests. Amplitude Experiment uses the means, variances, and exposures of your control and variants to forecast expected behavior and calculate how many days your experiment will take to reach statistical significance. As Amplitude Experiment receives more data over time, this prediction will improve. However, if any of these inputs change significantly during the experiment, the accuracy of the prediction will likely decrease.

Data quality guardrails

There are few more important rules of data analysis than Twyman's law, which is premised on the principle that the more unusual or unexpected the data, the more likely they are to be the result of an error in data measurement or analysis. You could apply the same tenet to experiment results. In other words, if a result looks too good (or striking) to be true, it probably is.

That's why Amplitude Experiment comes with built-in guardrails to ensure the quality of your testing data. These automated checks appear in a consolidated list in the Analyze tab, alerting you of any data issues that can lead to skewed or confusing results. They also show common pitfalls to watch out for in setting up and implementing experiments—and guidance for overcoming them.

Automatic checks include:

- Consistent number and definition of variants
- Consistent allocation between treatment and control variants
- If a test has sample ratio mismatches
- If a test has exposure events without assignment events
- If variant jumping is occurring at a high frequency
- Traffic decreases over time
- If there are suspiciously large uplifts and aberrations in event data
- If there are abnormal variance, standard error, or confidence intervals
- Exposures without assignments

Exposures without assignments

The [Exposures without Assignments](#) chart (in the Monitor tab) queries for the cumulative number of unique users who have performed an exposure event without a corresponding assignment event within each day. If you see a large number or percentage of users in the chart, be careful when interpreting the results of your experiment. Investigate what happens if a user inadvertently gets exposed to the experiment. Was the experience bad? Can the user even see the experience? What does it mean if a user sees more than one experiment when they're mutually exclusive? Exposure without assignment may also affect future experiments, so you should investigate and fix the issue.



Sample ratio mismatch

Amplitude Experiment also checks for [sample ratio mismatch \(SRM\)](#) issues. (To view any issues detected, click on Implementation & Instrumentation in the Analyze tab of the data quality guide). An SRM occurs when the observed allocation for variants significantly differs from the specified allocation. For example, imagine you've set your experiment's traffic allocation to be split equally between the control and treatment variants, but instead, the control receives 55% of the experiment's traffic. SRMs point to biases in the data and can lead to unexpected or inaccurate results if unresolved. Generally, you should be wary of the results of any experiment affected by a SRM. The cumulative assignment or exposure charts can help you track down the cause of an SRM. Look for timestamps where the control and treatment time series diverge; often, you'll find the cause there.

Variant jumping

In some cases, SRMs are caused by variant jumping. This is when the same user sees two or more variants for a single flag or experiment, which sometimes occurs with authentication patterns that make it difficult to know if a user has already been assigned a variant. Examples include applications with short-lived sessions and applications with large numbers of anonymous users.

Variant jumping may occur normally or abnormally, for various reasons, but variant jumping above a certain threshold may be cause for concern when it comes to an accurate analysis. Amplitude Experiment helps you debug variant jumping by flagging users who have jumped variants

so you can analyze their timelines. If you're using remote evaluation, you should check the assignment event to identify assignment vs. exposure discrepancies.

A rigorous approach to experimentation

When it comes to experimentation in your digital product, testing methodology is essential. A platform informed by best practices for experimentation and statistics can ensure you're doing it right. Amplitude's end-to-end platform was designed for the most reliable results, while providing a seamless, consistent experience for your customers.

SRMs point to biases in the data and can lead to unexpected or inaccurate results if unresolved.

Getting Started with Experimentation

Cultivating a culture of experimentation is the first step to building a product-led experimentation program—creating and executing a strategic experimentation system is the next. Learn more by applying for the [Reforge Experimentation + Testing program](#).

Using the best practices throughout this guide will set your experimentation program up for long-term scale and success. Your product and engineering teams will be able to confidently make product bets and release new features with measurable impact every time you ship. And your entire organization will benefit from data-driven decisions that move the needle on business goals.

Whether you're starting from scratch or scaling an existing program, [Amplitude Experiment](#) will take you to the next level. With built-in identity resolution, feature management, and statistical rigor, Experiment will guide you to results you can trust. Start building confidently today and [request an Experiment demo](#).



Contributors



SALEEM MALKANA | Executive in Residence - Reforge

Saleem is an Executive in Residence at Reforge. He is the former VP Product for NBC News, MSNBC, and other brands that together reach 120M+ users monthly. Previously, Saleem led Product at AMC Networks' DTC startup, growing subscription video brands from proof of concept to over 1M paying subscribers. He also brings perspective from his experiences at Amazon, MTV, and the CIA.



BHAVIK PATEL | Founder and Managing Director, CAUSL

Bhav is the founder of CAUSL, a Product Measurement Consultancy that helps product organizations connect the dots between their product initiatives and company goals. Bhav has previously held "Director and Head of Product Analytics" roles at companies like Gousto, MOO, PhotoBox, and most recently, Hopin. Bhav also runs London's biggest conversion rate optimization, analytics, and product meetup called CRAP Talks.



ELENA VERNA | Growth Advisor, Interim Growth Exec

Elena was previously Head of Growth at Amplitude. She is a growth hobbyist, helping companies build product-led growth models. She is a Program Partner at Reforge, Board Member at Netlify, and Advisor to Clockwise, SimilarWeb, and Veed. Previously, she was SVP of Growth at SurveyMonkey and CMO at Miro.



WIL PONG | Head of Product, Experiment, Amplitude

Wil Pong is the Head of Product for Amplitude Experiment. Previously, he was the Director of Product for the Box Developer Platform and Product Lead for the LinkedIn Talent Hub.



CHAD SANDERSON | Head of Data, Data Contracts Advocate

Chad Sanderson is passionate about data quality, and fixing the muddy relationship between data producers and consumers. He is a former Head of Data at Convoy, a LinkedIn writer, and a published author. He lives in Seattle, Washington, and operates the Data Quality Camp Slack group.



AKHIL PRAKASH | Senior Machine Learning Scientist, Amplitude

Akhil is a Senior ML Scientist at Amplitude. He focuses on using statistics and machine learning to bring product insights to the Experiment product.



KATHY QIAN | Product Manager, Amplitude

Kathy Qian is a data scientist turned product manager working on the Experiment product. She has an extensive background in both product and marketing experimentation and has helped multiple Fortune 1000 companies across the retail, grocery, hospitality, and restaurant industries design and analyze transaction, customer, store, and market-level initiatives.



AUDREY XU | Solutions Consultant, Amplitude

Audrey Xu is a Solutions Consultant at Amplitude, working with companies to uncover areas of opportunity and build better products. She is a self-proclaimed Amplitude nerd and graduated with a degree from U.C. Berkeley.

Start building confidently today and request an
Experiment demo →

About Amplitude

Amplitude is a leading digital analytics platform that helps companies unlock the power of their products. Almost 2,000 customers, including Atlassian, Jersey Mike's, NBCUniversal, Shopify, and Under Armour, rely on Amplitude to gain self-service visibility into the entire customer journey. Amplitude guides companies every step of the way as they capture data they can trust, uncover clear insights about customer behavior, and take faster action. When teams understand how people are using their products, they can deliver better product experiences that drive growth. Amplitude is the best-in-class analytics solution for product, data, and marketing teams, ranked #1 in multiple categories in G2's 2023 Winter Report.

