Optimizely

GETTING STARTED WITH PRODUCT EXPERIMENTATION

How Product Development Teams Reduce Guesswork and Innovate Faster

KEY DISCUSSION POINTS:

01 How experimentation fits in with other initiatives at your company, such as Agile

02 Best practices for integrating experimentation into your release process

03 Key considerations when buying an experimentation solution

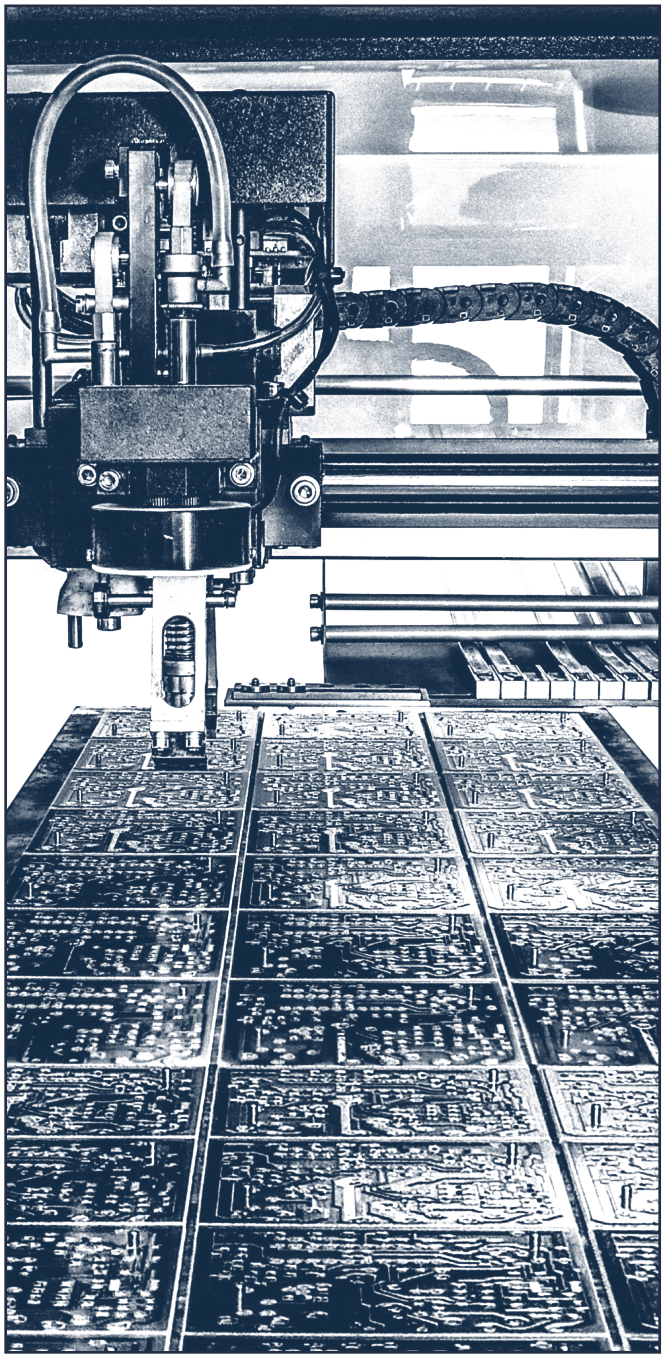
04 Ways to optimize performance and ensure security and compliance

CH-01

MOVING FROM AGILE TO EXPERIMENTATION-DRIVEN DEVELOPMENT

CH-01 MOVING FROM AGILE TO EXPERIMENTATION-DRIVEN DEVELOPMENT

DURING THE PAST 20 YEARS, THE PROCESS OF DEVELOPING AND RELEASING SOFTWARE HAS UNDERGONE A SIGNIFICANT EVOLUTION.



Many companies are moving beyond the lengthy, top-down release cycles driven by boxed software to an iterative release process driven by the rapid innovation of software running on internet-connected devices. Teams are embracing principles such as Agile, Lean, DevOps, and Continuous Development to ship quality software faster. In fact, Agile is now the norm: **94%** of companies report that all or some of their development teams currently use Agile.

Many have also migrated from monolithic applications to microservices architectures. By breaking up applications into multiple services and shipping smaller changes more frequently, software teams mitigate risks and become more responsive to their customers' evolving requirements.

☑ THE ULTIMATE GOAL IS TO GET QUALITY PRODUCTS INTO CUSTOMERS' HANDS FASTER.

The resulting feedback helps teams learn and iterate on their designs more rapidly. By incorporating automated testing along with Agile and DevOps, teams already accelerate development, uncover product bugs, and leverage opportunities to improve performance. Yet, no matter how reliable and performant software is, it must also deliver real value to end users and customers faster and better than any other product. That's where experimentation driven-product development comes in.

EXPERIMENTATION BECOMES THE FEEDBACK LOOP THROUGH WHICH TEAMS DETERMINE IF THEIR PRODUCTS ARE IMPACTING CUSTOMERS IN A POSITIVE WAY.

Put simply, experimentation helps development teams determine if they're building the right product. By adopting experimentation, teams eliminate uncertainty and guesswork from their software development process. By exposing new versions of software to a portion of customers while the remainder continue with the original experience, teams can quickly assess if their new updates are improvements or regressions. Experimentation, in essence, becomes the feedback loop through which teams determine if their products are impacting customers in a positive way.

Companies that adopt agile development and build out robust testing and QA processes, yet neglect to experiment at scale, are not getting the most out of their software development process. To perform at a high level, software creators can't simply optimize the performance and reliability of their products. They must optimize the actual benefits of their software as well. Without sufficient experimentation, however, they won't have adequate information at hand to make the best product decisions possible. The good news is, every software organization—including those now A/B testing in an ad hoc manner—can take key steps to expand their experimentation capabilities and strengthen their software development process.

Settling Debates With Experimentation

Often, smart people on product development teams will have differing opinions on the value of a new feature, or on which version of that feature will be most impactful. Consequently, teams will launch features based on intuition. The trouble is, common sense solutions aren't always the best ones. Even well-researched products can suffer due to the gap between what customers think they want and what their behaviors reveal they actually want.

Here's one example. Tripping.com, a metasearch site for vacation rentals, embraces experimentation-driven product development. Customers search rentals on their site, then they click through to third-party sites to make purchases.

Tripping.com hypothesized that by adding an intermediary listing page with more content, they could generate more revenue. This would be a counterintuitive change, however, since it would add friction to the conversion process. Nevertheless, they ran an experiment to see if it would impact bookings.

While the experiment showed a 3% decrease in clicks to 3rd-party sites, it also resulted in a 5% increase in revenue. The experiment confirmed their hypothesis. **Without experimentation, it's unlikely their team would have been willing to bet on this hunch.**

Some of the most revolutionary new products and features were originally controversial, or counterintuitive. Experimentation transforms the need for debate into an opportunity for discovery. With it, teams can now evaluate ideas and make decisions based on real, usable data.

When Optimizely founder Dan Siroker was a new project manager at Google, a mentor of his offered this advice on a controversial product launch: **"Don't frame it as a product launch. Just frame it as an experiment."** Once he did that, he experienced far less resistance to his idea.

☑ EVEN WELL-RESEARCHED PRODUCTS CAN SUFFER DUE TO THE GAP BETWEEN WHAT CUSTOMERS THINK THEY WANT AND WHAT THEIR BEHAVIORS REVEAL THEY ACTUALLY WANT.

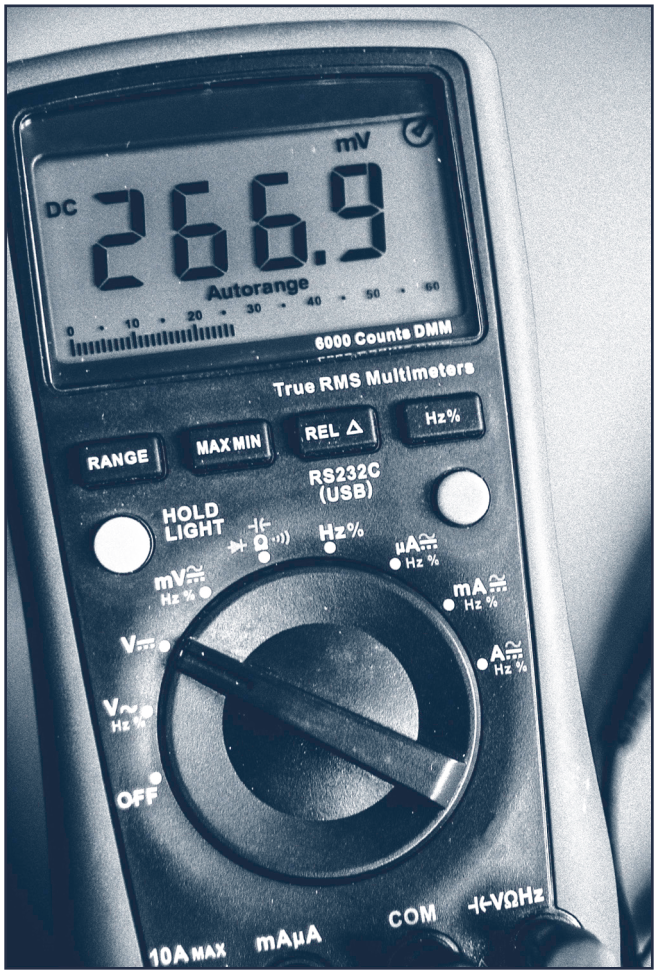
CH-02

PUTTING EXPERIMENTATION AT THE CORE OF YOUR RELEASE PROCESS

CH-02 PUTTING EXPERIMENTATION AT THE CORE OF YOUR RELEASE PROCESS

Integrating experimentation into the software development process helps teams launch better products. Industry leaders like **Google**, **Facebook**, **Microsoft**, and **Amazon** all run thousands of experiments each year. They've scaled and managed their businesses, constantly adapting to challenges from new entrants, by using experimentation as a core part of their product development process. Today's software development teams can benefit from implementing the best practices employed by these powerhouses. In this section, we discuss three key principles for making experimentation a core part of the release process.

☑ THE "NORTH STAR METRIC" (NSM) IS THE SINGLE MOST IMPORTANT METRIC THAT CAPTURES THE VALUE YOU DELIVER TO YOUR CUSTOMERS.



Principle 1: Choose the right metrics

The most important step before launching an experiment is agreeing on the metrics to measure. Metrics allow teams to define the success of their experiments. Typical metrics differ by industry, but are often based on conversion, retention, or lifetime value. Businesses with longer conversion cycles and higher customer LTV—mortgage lenders or furniture retailers, for example—will likely want to choose some metrics that are measurable in real-time, and some that indicate longer-term success.

Another essential step is to zero in a **North Star Metric** (NSM)—the single metric that best captures the value a company delivers to its customers. Using product analytics and research is the best way to determine what input metrics affect a company's NSM.

Take Postmates, a restaurant delivery service, for example. Their North Star Metric is the number of daily meals delivered. Postmates might establish that a metric higher in the funnel, like restaurant page views, is an indicator of daily deliveries. They may also determine that the number of page views is affected by the quality of search results, with no results being a worst-case scenario for a customer. With that in mind, Postmates could run an experiment to show additional restaurant recommendations whenever a customer's search comes up empty. The company could then measure whether this experience results in more restaurant page views, and ultimately, more meals delivered--the North Star Metric.

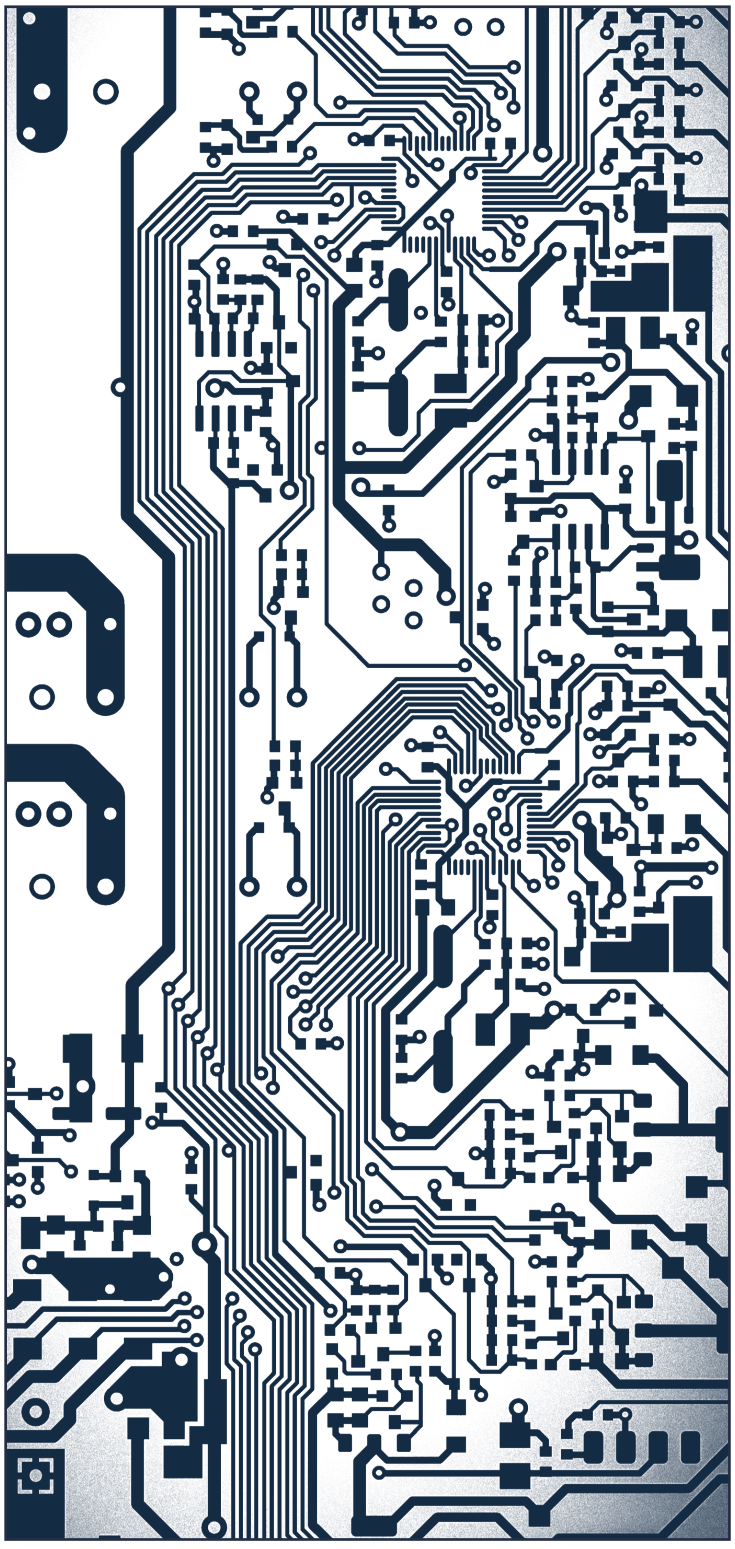
In the end, it's vitally important for every team member across product, analytics, and engineering to be in agreement about the team's core metrics for success.

Principle 2:
Your feature roadmap is your testing roadmap

Development teams frequently mention that they have difficulty thinking up ideas for experiments. They feel they would do better if they had an experimentation roadmap. But a special roadmap isn't necessarily the answer. In truth, a team's feature roadmap can and should be its experimentation roadmap. This is precisely what is meant by "experimentation-driven product development." A core part of every sprint process should be to decide when to attach experiments to new features and feature updates. For riskier features, it's helpful to experiment on a small portion of users before rolling the feature out for the entire customer base.

Leading software companies almost always test new features on a portion of their audience so they can assess its performance before launching it. Facebook, for instance, commonly launches new products to a small percentage of users in remote countries such as New Zealand. They then measure the launch's impact on core metrics before scaling to additional markets. By starting small, they learn quickly without risking a major backlash from a broader customer base. Whenever possible, software teams should launch features using a similar approach.

FOR RISKIER FEATURES, START BY EXPERIMENTING ON A SMALL PORTION OF YOUR AUDIENCE BEFORE ROLLING OUT TO 100% OF PRODUCTION.



PRIORITIZATION

Most teams are constrained by resources and can't experiment on every feature. When deciding which features to experiment on, consider a few factors. First, what is the risk to business metrics if an experiment degrades the experience? Second, which features are likely to have the greatest impact? Third, what is the level of effort required to implement a feature or experiment? In addition, larger, customer-facing updates should always be rolled out as experiments to first assess the impact.

DECISION-MAKING

Once an experiment is launched, a decision must be made at some point to either continue the experiment or to conclude it. A robust experimentation platform will help teams determine two things—the point at which an experiment reaches statistical significance and the variation that achieves the best results. In many cases, the original version remains the best option. Yet without first testing new versions, teams can unintentionally worsen the customer experience or negatively impact key metrics. After finding a successful new variation, you may be able to immediately flip a switch that directs all traffic to that variation. In the long run, however, it's ideal to fine tune the software by removing all experiment and variation level code. In the long run you'll likely want to clean up your code and remove experiment and variation level code.

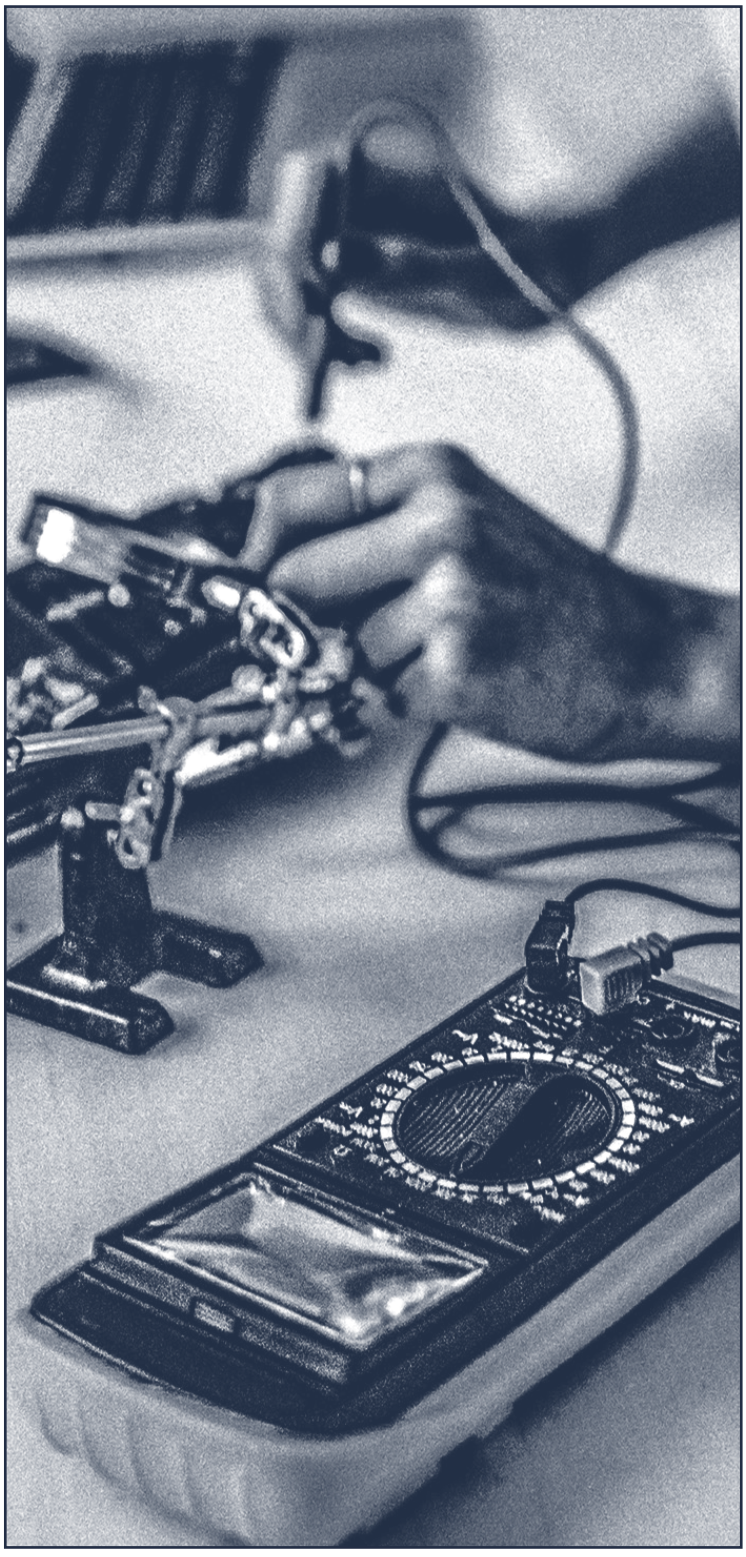
Addressing technical debt

Experimentation typically involves inserting conditionals into the codebase. This can result in technical debt. To mitigate this debt, teams should implement a regular interval during which they clean up the code generated by past experiments. One easy way to do this is to insert a story into the sprint every month, or every quarter, for this specific purpose. To make experiments even more impactful, it helps to rerun past variations to confirm that current ones still perform the best.

Principle 3:
Test Small, Test Often

Many software teams are either moving to, or have already moved to, a faster release cycle. Some have recently adopted Agile and are ramping up their procedures. Others are currently experimenting with ongoing software development. Regardless, these progressive teams are releasing smaller, incremental software iterations at a more frequent pace. While it's useful to have some bigger, riskier experiments in the pipeline, running successive, smaller experiments helps diversify risk. Plus, these small bets sometimes pay huge dividends. Microsoft's Hotmail team is one example of this approach. They conducted a seemingly small test in the UK to determine whether opening Hotmail in a new tab would affect user engagement. It did. The experiment resulted in a nearly 9% increase in people opening Hotmail. This same test was later replicated in the US for MSN search results. That test led to a 5% increase in clicks per user. Small tests can yield big gains.

☑ BEST-IN-CLASS TEAMS ARE MOVING TO SMALLER, INCREMENTAL BUILDS MORE FREQUENTLY AND YOUR TEAM SHOULD ADOPT THE SAME APPROACH WITH EXPERIMENTATION.



CH—03

CHOOSING THE RIGHT EXPERIMENTATION PLATFORM

CH–03 CHOOSING THE RIGHT EXPERIMENTATION PLATFORM

After buying into the value of experimentation, the next question is which experimentation model to pursue. Software teams can generally choose from three types of solutions. The first is to build a proprietary experimentation system from scratch. The second is to modify an existing open source framework. The third option is to purchase a software-as-a-service experimentation solution. This whitepaper doesn't address the pros and cons of building or buying an experimentation solution. Instead, it provides a brief primer on buying an experimentation solution. Those interested in learning about the tradeoffs between building an experimentation solution and buying one should read **"Implementing the Right Experimentation Solution: Choosing Whether to Build or Buy."**

Purchasing Experimentation as a Service

The greatest benefit of purchasing a solution vs. committing internal resources to building one is that it allows software developers to spend their time doing what they do best—creating great products for their customers. Many companies have moved from owning their own servers to adopting cloud services providers so they can focus more intently on their core business initiatives. Software companies are purchasing experimentation as a service for the same reason.

Typically, the first choice when purchasing an experimentation solution is to decide between a JavaScript-based visual editor solution that enables running client-side website experiments without code, or an SDK-based solution that enables server side testing and experimentation across multiple platforms, or a platform that enables both approaches.

Visual editor-based solution buyers should look for the ability to write custom HTML, CSS, and JavaScript code for more advanced testing. It's equally important for the solution to come with thorough documentation and additional developer controls like a REST API.

SDK-based solutions offer software development teams more flexibility, particularly those that feature the ability to experiment across, web, native mobile, and server platforms. Server-side experimentation lets teams go deeper. They can experiment on application logic layers to test the efficacy of new algorithms, complex redesigns, and even new features.

One critically overlooked component of experimentation platforms is the value of statistics and real-time analytics. Delays in getting data back from internal analytics teams can often create a bottleneck in the experimentation process. However, purchasing an experimentation solution with robust, built-in analytics typically accelerates experimentation velocity, enabling teams to run more tests. This feature won't take the place of an internal analytics team, but it will perform initial assessments and allow data scientists to concentrate on advanced analyses.

CH-04

ENSURE SECURITY AND PERFORMANCE

Don't Compromise on Performance

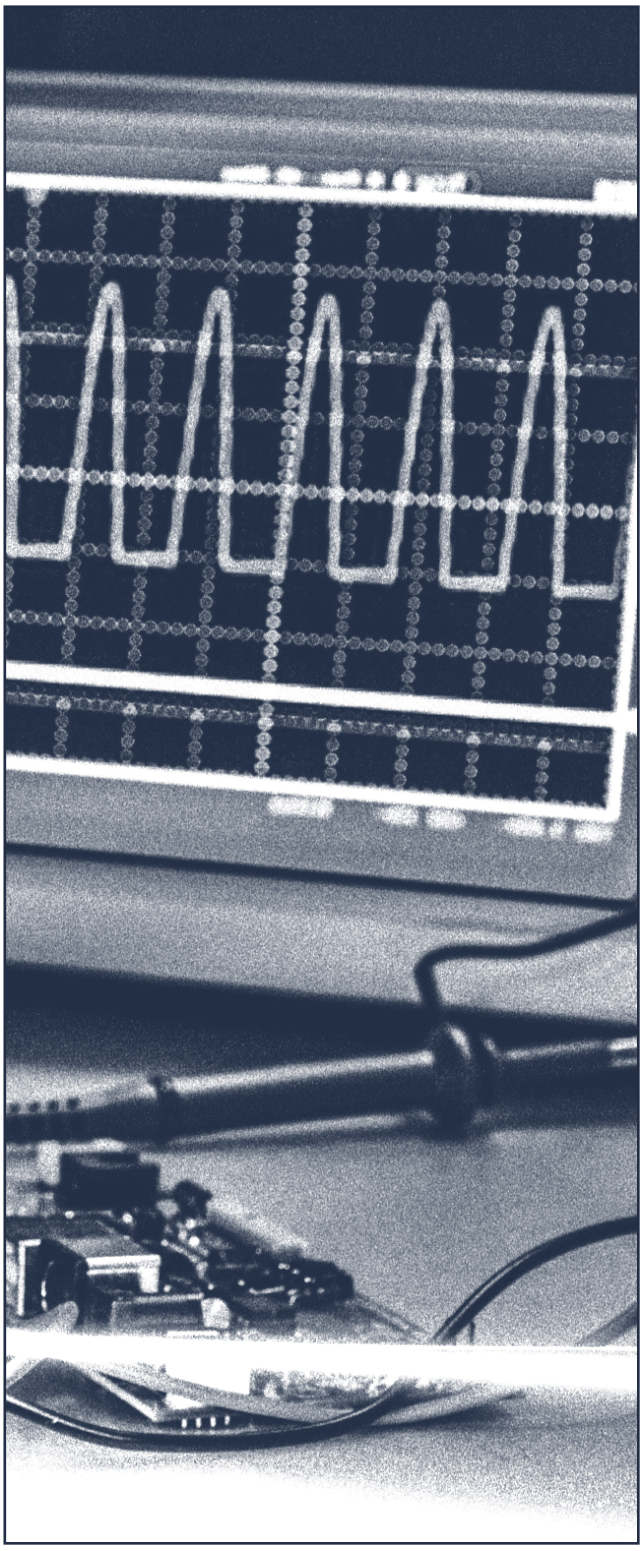
Developers are responsible for ensuring the products they build are secure and performant. Just as with any third-party technology, experimentation solutions present several critical security and performance considerations, especially for those deploying client-side experimentation solutions that include JavaScript snippets that modify the front-end experience.

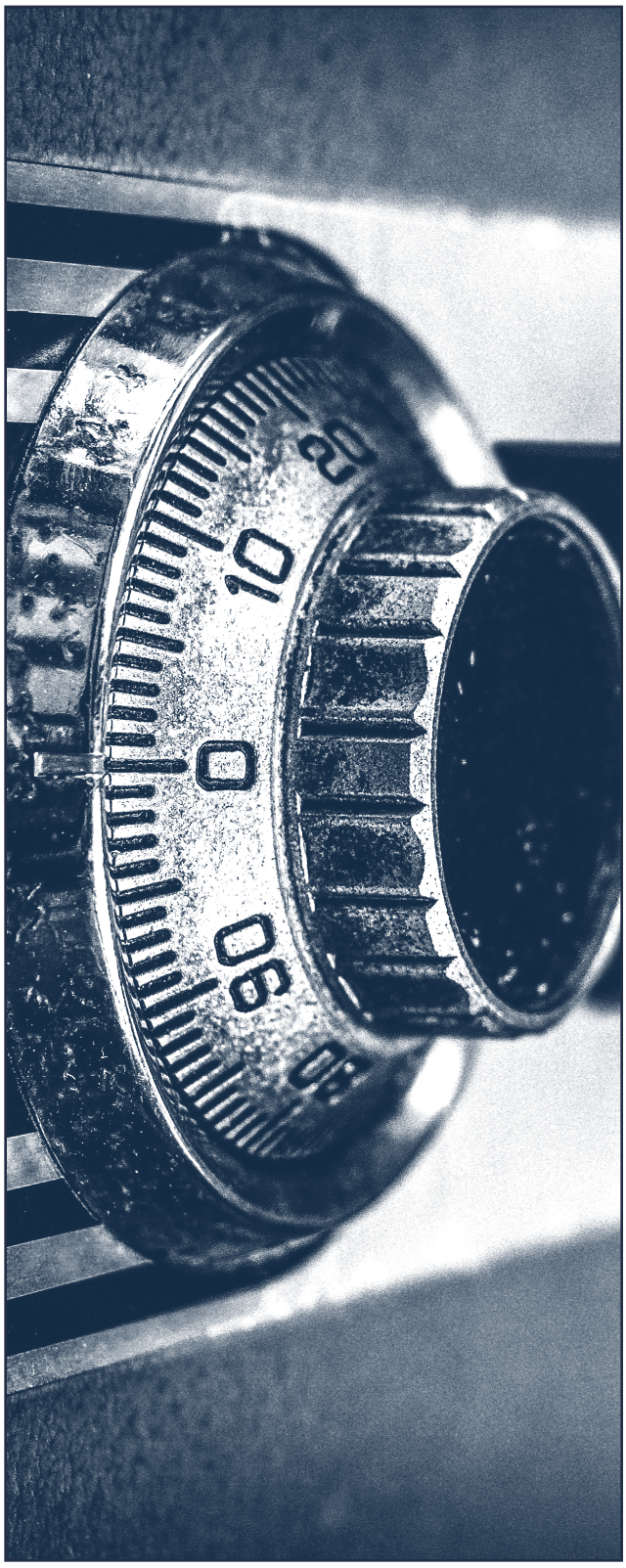
For client-side experimentation solutions, the biggest factors for performance are:

- ☑ **Placement of the snippet**
- ☑ **Size of the snippet**
- ☑ **Frequency of caching**
- ☑ **Number of events sent**
- ☑ **Timing of events sent**

Performance factors are snippet placement, snippet size, caching frequency, plus the number and timing of events sent. A robust experimentation solution will provide the controls necessary for managing such factors. When measuring performance, it's also important to focus on measuring real world performance such as time-to-paint metrics. Outdated metrics like page load time won't accurately represent a user's experience.

Conversely, server-side experimentation solutions typically have fewer performance concerns. Instead, it's wise to look for solutions that don't require API calls to handle traffic allocation and splitting, but instead make randomization and traffic splitting decisions in memory. Solutions that make API calls or leverage reverse proxy methodologies often experience issues under peak loads and deliver poor performance.





Ensuring Secure Experiments

Few things today are more vital that securing your technology. Client-side experimentation solutions should provide enterprise grade security along with compliance with standards such as ISO, SOC, PCI, and GDPR.

Key security features include:

- **TLS-Based authentication**
- **2-factor authentication**
- **Single-Sign On capabilities**
- **Encrypted communications**

For pages requiring the highest level of security—those with sensitive financial or personal information, such as checkout pages or credit or loan application pages—teams can use a PCI compliant client-side solution or use server-side testing.

CONCLUSION

Adopting any new system will require some learning and additional work. But the upside is undeniable. Embracing experimentation-driven product development will make your team more innovative, your business more agile, and your customers more satisfied.

About Optimizely

Optimizely is a digital experience optimization platform that enables product development teams to quickly experiment and roll out features across every platform.

VISIT OUR WEBSITE, AND DISCOVER THE MANY WAYS IN WHICH YOUR BUSINESS CAN BENEFIT FROM EXPERIMENTATION.