This is authors' version. The definitive version is published in the proceedings of ICSE'23 SEIP. ©2023 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

# A/B Integrations: 7 Lessons Learned from Enabling A/B testing as a Product Feature

Aleksander Fabijan
Microsoft
Bellevue, WA, USA
alfabija@microsoft.com

Pavel Dmitriev
Outreach.io
Seattle, WA, USA
pavel.dmitriev@outreach.io

Benjamin Arai
Microsoft
Bellevue, WA, USA
bearai@microsoft.com

Andy Drake
Microsoft
Bellevue, WA, USA
andrak@microsoft.com

Sebastian Kohlmeier
Microsoft
Bellevue, WA,USA
skohlmeier@microsoft.com

April Kwong
Microsoft
Bellevue, WA,USA
aprilk@microsoft.com

***Abstract*— A/B tests are the gold standard for evaluating product changes. At Microsoft, for example, we run tens of thousands of A/B tests every year to understand how users respond to new designs, new features, bug fixes, or any other ideas we might have on what will deliver value to users. In addition to testing product changes, however, A/B testing is starting to gain momentum as a differentiating feature of platforms or products whose primary purpose may not be A/B testing. As we describe in this paper, organizations such as Azure PlayFab and Outreach have integrated experimentation platforms and offer A/B testing to their customers as one of the many features in their product portfolio. In this paper and based on multiple-case studies, we present the lessons learned from enabling A/B integrations – integrating A/B testing into software products. We enrich each of the learnings with a motivating example, share the trade-offs made along this journey, and provide recommendations for practitioners. Our learnings are most applicable for engineering teams developing experimentation platforms, integrators considering embedding A/B testing into their products, and for researchers working in the A/B testing domain.**

***Keywords—A/B testing, A/B integrations, Platform design***

## I. INTRODUCTION

A/B testing enables companies to make trustworthy data-driven decisions at scale and has been a research area in the software industry for many years [1], [2]. Companies run A/B tests to assess ideas and to safely validate [3] what delivers value to their customers. For example, at Microsoft, we run tens of thousands of A/B tests yearly, testing the impact of UX improvements, infrastructure migrations, back-end service optimizations, and to determine the optimal time to update an operating system [4]. In our case study companies – Microsoft & Outreach – First Party A/B testing of the product (1P A/B testing) has been an integral part of software development process for years.

In addition to A/B testing for internal use, we have observed that A/B testing is becoming a differentiating feature [5] *for* software products themselves. In other words, not only are products using A/B testing to make informed ship decisions, but products are starting to *offer* A/B testing to their customers as one of their many features.

For example, Outreach is developing a sales engagement platform, a product that helps ease the communication between sellers and prospective customers. They offer A/B testing functionality for sales representatives to validate which sequences of e-mails and phone calls are most effective for different selling scenarios. Similarly, Azure PlayFab, which is a complete backend platform for live games with managed game services to test ideas for game features is offering A/B testing to game developers.

In this paper, we refer to these distinct use-cases of A/B testing as "A/B integrations", referring to A/B testing capability being integrated into a product as a feature for the customers as opposed to being used by the product creators themselves. As shown in Figure 1, we illustrate the conceptual difference between an A/B test for internal use (first-party A/B test e.g. testing a change in a product like Bing which has integrated with A/B testing platform), and A/B Integrations (e.g. testing a change in one of the games that have onboarded to Azure PlayFab which has integrated with ExP – Microsoft's experimentation platform).

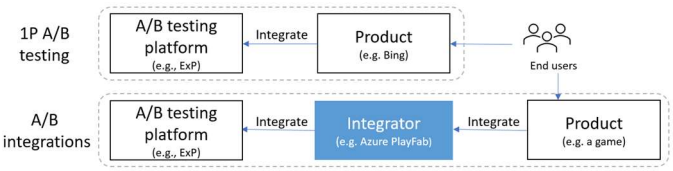


**Figure 1. 1P A/B Testing (top) vs. A/B Integrations (bottom).**

In this paper, based on a longitudinal multi-case study over the last four years, we present the unique engineering and cultural challenges that A/B testing platform teams needed to overcome to make A/B integrations successful. A/B integrations share some commonalities with first party A/B testing; however, they also differ in many aspects. Specifically, to offer A/B integrations, the A/B testing platform teams may be required to build new infrastructure, streamline metric creation, build

XXX-X-XXXX-XXXX-X/XX/$XX.00 ©20XX IEEE

partnerships with configuration providers, create education programs for integrators, and offer validation capabilities to assure trustworthiness.

Our main contribution is the discussion on the lessons learned which will be most helpful to those working on A/B testing platforms and those developing products that may benefit from A/B integrations. These roles typically include software engineers, product managers and data scientists, as well as researchers exploring A/B testing and data-driven software development.

## II. BACKGORUND

### A. Learning about user preferences

It is common for companies to use existing telemetry in products to understand what is valuable for the users and make decisions based on lessons learned [6]–[8]. Observational data such as telemetry [9] enable software companies to be more accurate in evaluating their ideas, and to move away from assumptions and towards trustworthy data-driven decision making through experimentation. In software development, the term "experimentation" can be used to describe different techniques for exploring the value of the changes introduced to a product [10]. For example, experimentation can refer to iterations with prototypes in the startup domain [11], [12], canary flying [13] of software features (exposing a small percentage of users to a new feature or a change), gradual rollout [13] (deploying a change to one customer groups and expanding to the next one), dark launches [14] (releasing new features disabled and testing them in production), and controlled experimentation [15] - releasing multiple variants of the product and evaluate the differences between them through statistical tests. We describe the latter next.

### B. First party A/B testing

In this paper, when we mention experimentation or A/B testing, we refer to the scientifically proven technique of randomized clinical trials [16] in an online setting, originating from the theory of controlled experiments going back to Sir Ronald A. Fisher's experiments at the Rothamsted Agricultural Experimental Station in England during the 1920s [15]. In the simplest A/B test, two or more variants of the product are tested against each other to determine, with statistical significance, if one of the variants performs better for key metrics. The tests are executed in parallel which helps exclude any other factors such as seasonality or special events like Olympic games from impacting the measurement. In this paper, we will use the term 1P A/B testing (first-party A/B testing) to refer to such experiments conducted in the product directly, and to differentiate it from the A/B integrations use case when A/B testing is offered as a feature of a product which primary purpose may not be A/B testing.

### C. Developing A/B testing capability

To run A/B tests at scale, companies need to invest in infrastructure, processes, and culture in an iterative fashion [17]. Prior work both from authors of this paper [18], [19] and from other researchers [20]–[22] and companies such as Booking.com [23], [24], LinkedIn [25], Google [26] and Facebook [14] stress that the growth of experimentation is conditional on the correct execution and integration of the scientific method into a software product. This includes developing an experimentation platform [19], designing comprehensive metrics for measuring the impact of A/B tests [27], and highlighting the value to create excitement and increase adoption of A/B testing [28]. All this research, however, has been focusing on building A/B testing capabilities and using them directly in a product. To our knowledge, no research body exists on how to build and integrate A/B testing capabilities into products such that products themselves can offer A/B testing as a feature to their users. This is a problem that our case companies, which we will introduce next, have had to solve to create differentiating capabilities in their software products.

## III. RESEARCH METHOD

In this section we introduce our case companies and describe how we collected and analyzed the data for this research.

**Case companies.** Two companies participated in our study, Microsoft, and Outreach. Microsoft is a large-scale software company with many diverse products that are running A/B tests [2] for many years. At Microsoft, over tens of thousands of A/B tests are run every year across the products on the web, client applications infrastructure, user experience, etc. using the experimentation platform ExP where authors of this paper are working at – taking the A/B testing platform team perspective. ExP team collaborated with many integrators. One of them which we can present in this paper is Azure Playfab [29], a gaming back-end solution that aimed to improve its A/B testing offering.

Outreach is a startup in the sales domain, providing a sales engagement platform for B2B sales. They have embarked on the journey of A/B testing in 2018 when they ran their first A/B tests. One of the authors of this paper works at the data science team at Outreach that provides A/B testing capabilities to Outreach customers – taking the A/B testing integrator perspective. Outreach has integrated A/B testing into their product by purchasing an A/B testing platform available on the market.

**Data collection and analysis.** The authors of this paper work *as subject matter experts* for scaling A/B testing in their companies and have collected data through action research [30]. Aleksander is a Senior Product Manager at Microsoft ExP where he works with Andy who is a Principal Data Scientist, and Benjamin who is a Group Partner Data Scientist and Product Manager on the ExP team. Sebastian is a Principal Product Manager and April is a Principal Software Architect on the ExP team. Pavel is the Vice President of Data Science at Outreach.io. In aggregate, the authors have over 50 years of experience in the field of A/B testing.

Our data collection consisted of several qualitative and quantitive data colleciton techniques. During the last four years, the authors of this paper have been working with software product teams and enabled A/B integrations for several products – Outreach and Azure PlayFab being two examples. This involved semi-structured interviews through which we collected requirements for platform features for A/B integrations, focus groups with engineers, data scientists and executives, longitudinal evaluation of the progress and surfacing of the blockers. We analyzed our collected data in 6 joint workshops

conducted over Microsoft Teams, each 1h in duration, where we thematically coded the observations and aggregated them into 7 key categories, which we describe as the main contribution of this paper. We resolved disagreements as comments on content.

**Threats to validity.** There are several validity concerns that are applicable for this type of qualitative research [31]. With respect to construct validity, researchers and participants in this study work in the field of A/B testing and were well aligned on the studied phenomena. In each data collection session, we explained the purpose and terminology at the beginning to all participants. With respect to *external validity*, the results of this paper apply specifically to teams in software companies that are developing A/B testing capability and consider offering this capability as a product feature. Specifically, the findings will help them understand the key requirements / effort needed to move from offering A/B testing as an internal service towards offering A/B testing as a feature. The learnings can be useful also to products that are evaluating A/B testing vendors/support (customer perspective).

## IV. INTRODUCING A/B INTEGRATIONS

In this section we first introduce A/B integrations. Next, we describe the key lessons learned that we obtained from enabling A/B integrations at our case companies. We conclude the section with a summary.

### *A. A/B Integrations*

As described in the introduction, companies have been A/B testing their products for many years. However, we have observed a unique shift in how A/B testing is becoming offered at our case companies. In addition to being a tool to decide what features to ship for a product being developed by our case companies, it is becoming a feature of a product for our customers. The main rationale for offering A/B testing as a feature is its capability to offer trustworthy evaluation. Specifically, by offering the capability to compare ideas for customers creating video games, web sites, e-mails etc., our customers can now test their ideas with their products.

To make A/B testing possible as an integrator feature, however, three parties need to be involved in the process from the software and product development perspective: First, there is the **A/B testing** platform **team**, developing the core platform and service for A/B testing. In our case companies, this team consists of data scientists, engineers, and product managers that are domain experts in A/B testing. At Microsoft, for example, this is the ExP team. Second, we have the **A/B integrators**: product teams that are integrating A/B testing into a software product. A/B integrator teams that our case companies partnered with typically consisted of engineers and product managers, and these teams are usually not domain experts in A/B testing. Finally, we have the **customers of integrators** – the users of the product developed by the A/B integrator. These users typically have specific tasks to complete using the integrator product (e.g. wish to schedule a meeting with a promising lead) and are the ones that will be using the A/B testing capability to validate ideas with their users.

Now that we are familiar with the main parties involved in the process, we will share the 7 lessons that we learned in how to make A/B integrations effective. We ranked them in importance based on the frequency count of how often they appeared in our thematic coding.

### *B. API/SDK infrastructure for A/B integration*

**Context & Motivating example**. In 1P A/B testing, an intuitive User Interface (UI) is the interaction point with the experimentation platform [32]. For example, experimenters use a web browser to start/stop A/B tests and analyze results. Interactions through the Application Interfaces (APIs), however, are secondary in importance compared to UI. At Microsoft ExP, thousands of unique users interact with the A/B testing platform daily using the UI while distinct API calls by users or applications were only a small fraction of this base before A/B integrations. And while high-quality API infrastructure is important for both 1P A/B testing as well as for A/B Integrations, the latter require a significantly broader set of features to be built in the API infrastructure. The two main reasons for this are the need for automation and debuggability. Specifically, in the case of A/B Integrations, there cannot be any human involvement or manual steps in A/B testing workflows – e.g. A/B tests are created and analyzed through the integrator product which is relying on the API infrastructure to execute operations in the A/B testing platform. In the 1P A/B testing, the user of the A/B testing platform can open a support case to get assistance with an issue. In A/B integrations, the integrator must have all the necessary information to help the customer. As a result of this additional distance, the API infrastructure needs to provide A/B integrators with additional debuggability on the status of the tasks, their results, and errors.

At Outreach, for example, the A/B testing vendor they integrated with was unable to supply a library to reproduce A/B test assignment offline. The lack of this capability prevented Outreach from using common A/B testing features such as retrospective AA [33] and required them to derive different ways to confirm experiment randomization quality. Similarly, at Microsoft ExP, API infrastructure for interacting with the experimentation platform has been in use for over a decade. However, to enable A/B integrations, the case company had to rebuild the API infrastructure to meet the needs of A/B integrators. The team had to expose signals that report when experiments results are ready, provide additional diagnostic information for failures, disruptions or delays, expose experiment quality concerns in the API such as Sample Ratio Mismatch [34], and provide guidance to A/B integrators such that they can create features in their product (e.g. UX for surfacing the quality issue) or set-up internal protocols (e.g. scaling compute resources in case of high capacity utilization). This required updating the contract of the APIs, exposing operational telemetry, adding new features, and supporting event-driven architecture. Even with ExP's mature A/B testing infrastructure built over 15 years, it took more than 12 months of engineering effort to support A/B Integrations. Furthermore, volume of traffic is another dimension - Our data shows that A/B Integrators may call APIs/SDK infrastructure with 100x larger traffic compared to 1P A/B testing users over very short period of time. To handle this volume, throttling and caching

needed to be added across API surfaces, and event-driven architecture is helping distribute the load more evenly.

**Explanation**. There are two main reasons why API infrastructure will require an additional investment for A/B integrations. **First**, and as illustrated with examples above, A/B integrators and their customers require a higher level of automation and debuggability. A/B testing platforms therefore need to provide a set of APIs or API attributes specifically for understanding the trustworthiness of A/B tests. We recommend supporting event-driven architecture such that integrators can automate tasks for important milestones like experiment results ready, experiment results invalid, or experiment running for a longer than expected period. Simply exposing a subset of private APIs to the integrator is insufficient.

Second, each integrator may use a different language (e.g. C#, Python) for integrating with the experimentation platform. This necessitates the need for supporting language-agnostic API infrastructure through a single specification which can be used to support the automation of SDK generation for all languages.

**Recommendation**: To enable A/B testing for integrators, expect to invest in automation and debuggability as part of the core API/SDK infrastructure. This investment will be larger and broader compared to the offering in internal/service-to-service APIs. Next, supporting several programming languages will quickly become a requirement even if it may not seem like one in the beginning. Therefore, start with automation that includes auto generating the API/SDK code from a single specification source from the beginning. This will make it easier to create compilers for auto-generating code in a different language in the future when the need arises.

Furthermore, define or select API guidelines early in the engineering process and establish a review cadence that will enforce following the guidelines. At Microsoft ExP, the vNext API design guidelines were published and made available on Github whereas other case companies followed their own internal guidelines. Most importantly, perform customer development throughout the development and identify champion scenarios that will be used by multiple A/B integrators. For these scenarios, create example code on how to use the APIs/SDK and share them with the integrators for a quick ramp-up. At Microsoft ExP, the team created 12 pages of champion scenarios with example code, covering the critical scenarios of editing an A/B test, starting / advancing experiments, programmatically editing metrics, scheduling compute jobs, retrieving A/B test results, and checking for validity of results.

The Champion scenarios contained guidance on how to utilize the built APIs for automation and debuggability. When the API and SDK infrastructure is ready, the platform team can be the first integrator to validate the infrastructure (dogfood). Use this opportunity to set a Key Result [35] to onboard any existing UX/tooling to the newer infrastructure and deprecate the prior APIs to save on the support costs. APIs that you create should seamlessly integrate with reusable UI components which we discuss in section F.

In Table 1, we summarize the common differences between 1P A/B testing and A/B integrations for this learning.

TABLE I. SUMMARY OF INFRASTRUCTURE LEARNINGS.

| Observation | 1P A/B testing | A/B Integrations |
|---|---|---|
| Primary interface between platform and user | User Interface (UI/UX) | API/SDK contract<br>Fully automatable onboarding |
| Use of API/SDK infrastructure | Mainly powering platform UI | Only form of integration |
| Customer familiarity with A/B testing | Low to Medium to high | None to low |
| Automation and debuggability | Low | High |
| Documentation | Intuitive UI | Champion code snippets |

### *C. Streamline metric editing experience for A/B integrators*

**Context & motivating example**. Designing good success and guardrail metrics is an active area of research [27], [36], [37]. The process requires taking raw telemetry emitted by a product as users are interacting with it and aggregating it into meaningful measures [9]. This typically involves many steps across multiple roles. For example, product experts and metric design researchers using the product and interviewing users to understand what constitutes success or dissatisfaction, engineers logging the telemetry that will be needed for data engineers to extract and transform the data [9], and data scientists creating the metric definitions and helping design A/B tests. For example, to measure the success of ideas tested on a search engine, one could take all the interactions with the product during the A/B test period and aggregate them by user to decide if one group had more positive experience on average than the other (e.g., fewer query reformulations per user). Designing metrics in this way is not optimal nor feasible for customers of A/B integrators.

**Explanation.** The ability to integrate with an existing A/B technology opens A/B testing to product teams who would not be willing to commit the resources to build their own experimentation solution or integrate with an external solution. The customers of integrators therefore need pre-defined metrics to simplify the process and get them value quickly. In our research, we observed that metric definitions and notion of success can be shared across customers of an A/B integrator. This is not always the case for 1P A/B testing, e.g. what makes users of Bing successful could vary significantly from what is considered success for users of Outlook. To provide an example, for every customer of Outreach – a sales engagement platform used by sales representatives to reach out to potential customers - it is important to measure, for every change to an email template, the sentiment of customer responses to the emails, and how many meetings that email template helped setup with potential customers. These success criteria are the same across Outreach customers. Similarly, customers of Azure PlayFab get the same starting core set of metrics. In addition, supporting custom metrics still remains a requirement, e.g. both of our case companies have received requests from their customers for them. However, by having a starter set of metrics for customers, metric creation no longer is a blocker for getting value for A/B, and instead becomes a value add the customers can leverage as appropriate.

**Recommendation.** The key advice here is to provide A/B integrators with the ability to easily create metrics that can be shared across their customers from day one. At one of our case companies, the A/B testing platform team created a solution to provide consistent metrics to every customer of an integrator. The solution involves adding the capability to create a centrally managed metric set that can easily be cloned for every new customer of the integrator, assuring correct isolation of resources and seamless management of metric definitions. Individual metrics are carefully tailored for integrators customer base and all of the customers can use the metric definitions for their A/B tests. This is different from 1P A/B testing where each product typically requires its own research and metric definition process.

To achieve this, it was critical to standardize the telemetry logging contract on the A/B integrator side, which needs to include the required parameters (e.g. variants the user has been randomized into). To standardize the logging, the platform team needs to provide the expected schema to the integrator and collaborate with the integrator on validation. Furthermore, if a customer of an A/B integrator can create new flows in their product, the A/B testing platform needs to be able to accept new event types and can create new metrics. This can be done by either defining new event names, or using property bags in the schema, and calling the API/SDK infrastructure described in the earlier section to create those new metrics.

### *D. Enable effective metric computation and analysis*

**Context & motivating example.** Outreach uses a 3rd party vendor to execute A/B tests for their customers – sales organizations. However, when it comes to computing metrics and performing statistical tests, Outreach faced a challenge. With different pieces of telemetry being spread across SQL databases, data lake, and their A/B testing vendor, there was no clean and cost-effective way to compute metrics. They would either have to send all their telemetry to the A/B testing vendor, which is concerning, or they would have to implement metric management and statistical analysis themselves – a difficult, costly, and error-prone task. While Outreach ended up doing the latter, they wish A/B testing vendors provided better options.

**Explanation.** Importance of leveraging a diverse set of metrics to analyze A/B tests and implementing statistical tests accurately has been strongly emphasized in A/B testing literature [38]. Companies that use a 1P A/B testing platform can easily share learnings and compute resources between the product being testing and the A/B testing platform, allowing A/B testing platform developers help implement best practices in an efficient and cost-effective manner. Integrators, however, face the following four challenges:

1. Integrators may already have telemetry stored in other types of stores for uses such a reporting and alerting. Indeed, the number and diversity of telemetry stores, ranging from relational databases to data warehouses, to data lakes, to analytical tools, across different clouds and on-prem deployments, make it impossible for an A/B testing platform to support all options and keep up with evolution.
2. For the same reasons as above, typically integrator's computation environment may not be natively supported by the A/B testing platform.
3. Setting up a separate storage and computation environment just for A/B testing would duplicate compute resources and likely be costly to support over time.
4. Integrators lack domain knowledge and resources to implement best practices of metric computation and statistical testing in-house.

From the integrator's perspective, the ideal solution should be to compute metrics on top of their data store in their computation environment, while the process is implemented and managed with the help of the A/B testing platform. The challenge for the A/B testing platform is how to enable such capability.

**Recommendation.** In recent years, semantic layer emerged as a solution to different metric computation needs [39], [40]. Semantic layer provides a way to define metrics in an abstract form, independent of specific data stores or computation environments. Semantic layer vendors then integrate with various data stores and computation platforms to auto-generate and execute metric computation code, as well as provide APIs to access metrics. Thus, semantic layer serves as a middle layer between data/compute environment and metric consumers and is used as a single source of truth for metric definitions across use cases such as internal dashboards, in-product analytics, and machine learning.

We recommend that A/B testing platforms integrate with semantic layer vendors. This approach has been successfully implemented within large companies such as Airbnb and Microsoft [41], [42]. While many 3rd party A/B testing vendors have not yet embraced this approach, based on the learnings that we have, we believe this is the best way forward for them as well.

Aside from resolving the 5 challenges mentioned above, this approach has several other advantages:

1. It allows automatically providing "default" metric set consisting of "count" and "boolean" metrics for each event of interest (e.g. Avg Number of Emails Sent per User, Fraction of Users who Sent at least One Email). This may be beneficial for integrators to get started on creating their own more nuanced metrics, ensures important events are not left unmonitored when analyzing test results, and helps automatically catch unexpected data quality issues.
2. It provides an easy way to compute and monitor a consistent set of metrics over time for both dashboards and A/B tests. These metrics would be used as success measures, helping align A/B testing success criteria and business KPIs.
3. It naturally supports the tradeoff between speed – how quickly the experiment results are available, and cost – how expensive it is to compute these results. Since semantic layer supports multiple computation platforms, a subset of A/B testing metrics can be configured to run faster (e.g. once every hour) on a more expensive

computation platform, while the full set of metrics can run less frequently (e.g. once a day) on a more cost-effective platform.

4. It provides more flexibility for defining the business model, where the price integrator pays for the A/B testing platform may depend not only on the number of experiments ran or traffic that passed through those experiments, but also on the number of metrics and metric computations performed as well as the complexity of those metrics.

### *E. Config service as integrator*

**Context & motivating example.** Configuration provides the ability to deliver metadata to applications in real-time, so that applications can change their behavior in response to this metadata [43]. This enables the application to perform code-less feature changes and turn features on/off in real-time. A/B testing and configuration are similar due to the ability to turn features on/off in real-time. Since product teams often already have a configuration system or there are central configuration systems that are used more broadly across an organization, part of Microsoft's ExP platform strategy is to pursue integration opportunities with config providers to help accelerate experimentation adoption. By integrating experimentation capabilities directly with configuration systems that are used across an organization, ExP can easily align with existing processes that teams have in place to deploy and manage general configuration. It also enables larger organizations that span 1,000s of engineers to standardize how feature flags are authored and managed to ensure consistency across the organization.

**Explanation.** One scenario where configuration leverages turning on/off features is the gradual rollout to users to manage risk (aka., blast radius). Similarly, A/B experiments turn features on/off for samples of users to determine impact of the feature being tested.

From a design perspective, it's easy to see how configuration and experimentation are tightly coupled and ideally a single system. By having a single system, multiple software development efficiencies can be achieved. For example, the A/B testing platform can provide a single user experience to rollout features and simultaneously generate A/B analysis to evaluate each step in the rollout process, A/B treatment assignment can leverage configuration metadata layer to deliver treatment assignment information to the customer, and developers need to integrate flagging a feature on/off. As a result, changes to application behavior are then controlled by a single system rather than multiple systems, making it easier to monitor and debug issues.

If a customer were to integrate with experimentation and configuration at the same time, it's clear that a single solution would be ideal. The challenge that the case companies ran into working with numerous customers is that customers tend to already have a configuration system in place. This can be an external system or a homegrown solution to meeting specific needs of a company's Safe Deployment Processes (SDP). In this case, the solution is not as simple. Convincing an organization to migrate to an all-in-one solution (viz., configuration + experimentation) is unlikely given it's likely deeply integrated with existing SDP processes.

**Recommendation.** For experimentation to work seamlessly with existing configuration systems, we have found it necessary to simplify the experimentation integration model. Specifically, it's critical that the A/B testing platforms enforce experimentation promises (e.g., trustworthy user randomization, computation of scorecards for analysis, consistency of users across rollout steps), but it is up to the configuration system to deliver the actual payload of assignment to the customer and in most cases provide the user experience to gradually rollout features and enforce policies to ensure features are rolled out in a consistent and safe manner (aka., manage blast radius). Integrating A/B testing into configuration management as a feature is an industry standard that has been adopted broadly by large software companies such as Amazon which has invested heavily in resilient continuous configuration [44]. They have found "that configuration changes cause outages at about the same rate as code changes". By having strong alignment between configuration management and A/B testing teams can reduce risks associated with configuration changes, make data-driven decisions and move faster with smaller and lower risk deployments.

The key to make integrations work with a myriad of config providers is to support different levels of config merging – specifically, make it possible to integrate A/B tests into standard config files by the integrator (e.g. the integrator merges the different flags being tested with config and delivers a single config file to the end product) as well as support the pass-through model where config and A/B flags are delivered separately to the product and the product is responsible to reconcile the two. For example, at our case organization Azure PlayFab, integration was a good option as Azure PlayFab already provided a config solution and event logging as part of its core offering. A/B flags were combined with the existing config delivery to remove any need to change game applications to get experimentation flags. This same approach, however, was not possible with some other integrators and a passthrough method was applied.

### *F. Reusing A/B testing platform UI*

**Context & motivating example.** When A/B testing is integrated, a UI is typically needed for the customers of integrators to use the functionality. The obvious choice here is for Integrator product teams to build UI for this purpose. However, in case of B2B partnerships there is an alternative – integrators can sometimes reuse A/B platforms UI and there are good reasons to do this. A/B testing platform teams have been publishing research on the importance of intuitive and comprehensive User Interface (UI) for the process of running and operating an A/B testing platform for many years [1], [2]. A/B testing UI can be as simple as a notebook with sample code on how to make an API call to start an A/B test for new teams starting to run A/B tests or a well-designed and comprehensive user experience. The UI at Microsoft ExP is created through the collaboration of a designer, UX Product Manager, UX Architect, and several UX-focused engineers and feature product managers working on the UI for the A/B testing platform. The ExP UX team performs user research to find pain points and for

designing new features, cognitive walkthroughs with various levels of mock-up maturity, A/B test our own ideas with platform users, and ship new features only if they don't regress our guardrail metrics.

Integrator products, however, don't necessarily operate in the A/B testing space. Furthermore, their UI principles may be fundamentally different. Ramping up on A/B testing UI can therefore be an expensive investment for an integrator in isolation. At Microsoft, for example, we have seen various levels of UI re-use for A/B testing among integrators. Specifically, Azure PlayFab is using ExPs UI for diagnostic purposes, however, their customers are using only Azure PlayFab UI and APIs to interact with A/B testing. So, how can the platform team enable integrators to be more effective in creating their UI and effectively grow their A/B testing offering?

**Explanation.** UI from the A/B testing platform can serve two purposes. First, it can be used to quickly ramp up the integrator team on A/B testing. Second, it can inspire how to build the A/B testing UI for Integrator customers or even be reused in the integrator product. As a result of the investment in ExP UI, the team often gets asked by integrators whether the UI can be re-used in the Integrator product. As a result of this common ask, the team has made it possible to reuse the UI at three different levels:

**Level 1**: **Design library.** ExP team has a shared library of designer created mock-ups for common A/B testing workflows. These include user stories with screen-by-screen frames. The design library consists of Fluent components – an open-source cross-platform design system that gives developers and designers the basic framework. https://www.microsoft.com/design/fluent/.

**Level 2**: **Component library.** ExP team has implemented common patterns from the design library as standard components in code. Standard components can be as small as drop-down and as large as a complete flow (e.g. complete page). An example of a large standard component is a multi-step wizard to create an A/B test with all the guidelines on how to integrate this wizard to create an entity. The components are implemented in React and include thorough accessibility checks (e.g. will work only with keyboard and well if used with screen readers), logging (e.g. will log interaction), are parametrized (e.g. can easily be re-used for other create flows) and have tests (e.g. standard CI/CD pipeline will validate the functionality). The team has created over 50 standard components for various parts of the UI. Standard components can be used across the A/B platform UI to make consistent experience, however, can also be used by integrators in their UI if they choose to do so.

**Level 3**: **Cross-linking to A/B testing platform**. Another way A/B testing UI can be reused in an integrator product is to expose links to A/B testing entities from the integrator product to an A/B testing platform. For example, once A/B tests results are available, the integrator exposes a hyperlink to the A/B testing platform where the results can be viewed. This will require the customers/users to authenticate into the A/B testing platform, however, if the alternative is no UI this is still better.

**Recommendation.** To ramp up the integrator team on A/B testing, we recommend identifying a specific integrator customer scenario, creating a proof-of-concept integration with the platform through the API infrastructure, and letting the integrator's product team use the A/B testing platform UI to set up and analyze an A/B test for the specific scenario. This will help them ramp up on A/B testing quickly as well as understand the appropriate level of UI reuse that will be applicable for them and their customers. The level does not need to be the same across the Integrator offering. For example, an integrator may choose to reuse design patterns for creating and editing A/B tests, reuse components from the code library for a different part of the product, and cross-link to A/B testing platform to view results. Each of the three levels of UI reuse has pros and cons that need to be considered when making a decision.

**Level 1: Design library approach** may result in significant savings for the Integrator design team, however, it will not have a high impact on the engineering efficiency. It is the easiest to share, however, the engineers may still need to implement the large majority of the components used in the design library. We recommend this approach for integrators with mature UI/UX teams.

**Level 2: Component library** approach may result in some of the UI in the integrator product looking or behaving slightly differently to the rest of the UI and has an onboarding cost for the Integrator team to learn the platform-team code base. It also creates a hard dependency on the specific A/B testing platform and changes that they may do to their UI. Once onboarded, however, the integrator team can create a new UI for the A/B testing functionality in their product quickly. We recommend this approach for integrators that are well connected with their A/B testing platform team as it will require a continuous partnership.

**Level 3: Cross-linking** approach may create some confusion as the customer of the integrator will have to learn two different UI experiences, however, is applicable as a stepping stone towards creating a comprehensive UI. If integrator customers are technical, however, developing additional UI may not be needed in the initial release of A/B testing as an integrator feature. This scenario is possible if A/B testing platform supports isolated environments with role-based access control.

At Microsoft, majority of internal-facing integrator products have chosen to re-use component library at some parts of their experimentation interface and cross-linking to ExP UX elsewhere. This combination and depending on how much of the UX a product team is comfortable to re-use, has significantly lowered the implementation effort for integrating A/B testing into a product. To give an example, one of the internal-facing integrator teams was able to implement an end-to-end proof of concept where a customer using their product was able to run an A/B test using ExPs infra in a few months of engineering effort. This work was sufficient to demo the new feature to their customers, shortening the engineering effort time to a Minimum Viable Product by at least 35% of the initial investment.

In contrast, external/public-facing integrators have chosen to use ExPs design library. Selecting this option resulted in fewer design and research studies.

### G. Support Education of A/B testing

**Context & motivating example.:** Even with amazing tools/SDKs to integrate and create high-quality metrics, deficiencies in educational tools will almost always result in a failure of A/B testing to scale and thrive within an organization. A/B test results are only as good as the telemetry and metrics that they measure. This can make A/B testing a time intensive endeavor, given the investment required. Education is a key tool to build momentum, and ensure teams are aware of the platform, key concepts of A/B testing, and best practices.

**Explanation**. Even with cutting edge tooling to shepherd customers through the A/B process, it is still susceptible to bad data (e.g., incorrect integration with treatment assignment, missing telemetry events) and interpretation of results (e.g., defining metrics using the wrong base level such as session or user [27]). Further compounding the problem, lack of familiarity with the platform can result in an integration that is trustworthy but inefficient and/or unable to service large volumes of experiments (e.g., each experiment requires computation over an unmanageably large amount of telemetry data). For A/B testing to scale and be cost effective for a business, there needs to be a way for a core set of people to horizontally educate the broader organization and in this case integrators on the value and democratize the data driven decision making process to help scale. Education is the key tool we have found to unlock these capabilities and increase the potential for customer success.

**Recommendations:** In our experience working with a variety of customers across Outreach and Microsoft, we have found the following educational paths critical for success and we recommend the following educational support tools:

1. Presentation with leadership to educate on the value and cost of experimentation. Value can be demonstrated with case studies and other motivating examples to justify business investment. Cost of experimentation is not a motivator but it is a critical component to educate on how experimentation requires a flywheel of investment [17].
2. Documentation to educate data engineers on the details and nuances associated with building trustworthy, scalable, and robust pipelines (e.g., high-volume telemetry should consider aggregation caches).
3. Documentation to educate developers on the appropriate ways to set up an experiment in code (e.g., upon failure the client should always be in a safe default state) and clean up (e.g., remove config data over time to manage risk) once the experiment is completed. We have found such documentation to be an important tool for developers to get hands-on experience integrating with the platform and delivering early milestones such as a proof-of-concept.
4. Classes to educate data scientists on the tooling available to assess the trustworthiness and interpretation of results from an A/B experiment. These tend to be delivered best as interactive sessions to allow the students to ask questions and understand the subtleties of A/B analysis.
5. Integrator documentation. Integrators will require adding documentation on how to use A/B testing and adjust it to use the language of their UI (e.g. see example from Azure Playfab [29]).

Armed with these educational tools and a strong champion (e.g., accountable architect or product manager), a team should be able to motivate A/B experimentation across all-levels of the business and execute on an integrating that is both trustworthy and scalable to meet business needs.

### H. Surface validation problems

**Context & motivating example.** A/B testing platforms need to provide integrators and their customers with tooling that enables trustworthiness validation during initial onboarding and continuous validation after completing the onboarding. When ExP onboarded Azure PlayFab as an integrator, the ExP and Azure Playfab teams collaborated closely to assess the trustworthiness of random assignment using an A/A test to demonstrate that there's no statistically significant difference in metrics. Both teams also engaged with an end-customer to pilot the integration with a test A/B experiment leveraging a pre-defined set of metrics. Post-launch from a continuous validation perspective, both teams collaborated to add a Sample Ratio Mismatch (SRM) column to Azure PlayFab scorecards to enable end-customers to assess the trustworthiness of their experiments in a self-service manner.

**Explanation.** Recent research contributions from large scale companies such as LinkedIn [45] Yahoo [26], and Microsoft [46] confirms that quality issues such as SRMs are common in large scale experimentation. At Outreach, before automated SRM validation was introduced, almost 50% of active experiments showed an SRM. After adopting SRM validation and alerting, the case company was able to reduce the SRM rate to single digit percentages, resulting in more accurate decisions and time savings on investigations and debugging. Validation leveraging an A/A test and a test A/B experiment is not just critical during initial onboarding of an integrator, but validation needs to also be easily repeatable as end-customers start running A/B tests. From that perspective it's critical for end-customers to be able to run an A/A test when needed and have visibility SRMs for debugging purposes. This information should be made available directly in the UI to ensure that users have visibility into potential trustworthiness issues and can investigate as needed.

From a metrics perspective, to keep things simple integrators are encouraged to provide end-customers with standardized definitions of metrics such as latency, system errors, user sign-ups and sales that can be easily leveraged by all customers. However, if integrators provide end-customers with the ability to create custom metrics it is critical that users have tooling to validate metric definitions and can easily assess health of their custom metrics. During custom metric creation it is recommended for the integrator to provide users with visibility into sample data to ensure that users can easily assess how a metric is computed. Once computed, A/B test scorecards should flag potential metric health issues such as missing telemetry, constant value and high variance and provide a UI that enables users to address issues. This includes supporting scenarios such as the deletion of a problematic metric or outlier trimming to improve metric quality.

**Recommendation.** When offering A/B testing as a feature, it's critical for integrators to provide a mechanism for one-off validation of assignment and continuous validation to assess the trustworthiness of scorecards and metric health. This needs to be fully automated to empower the end-customer by providing them with the visibility they need to address issues.

1. Integrator needs to choose how they use the APIs for validation (e.g. SRM). We recommend exposing the completion of a critical check as an event through an event-driven architecture and diagnostic information about the event through an API. This helps integrators call the API only when needed (e.g. for failed tests).
2. Platform should support multiple validation options, the most important are A/A tests and integrators can configure what and how often they will run them. Integrator then runs this test suite. AA tests, SRM tests, and other "black box" validation techniques for auto-detecting bad experiments are important, since human supported validation is not available nor scales for A/B integrations.
3. Self-validating platform. Platform should have validations built in and continuously running. More generally, things common across all integrators should be automated by the platform. It is hard for integrator to perform validations with no access to the platform.
4. Platform needs to expose logs to integrators. An integrator then decides how to surface those further. We recommend event-driven architecture for surfacing important events such as a test failing and providing integrators with guidance on what to recommend to their customers once one of such validations is unsuccessful.
5. While the above recommendations make it easier to diagnose quality issues with A/B tests, understanding the root causes behind those issues and fixing them is still a difficult problem [47]. It is important that the platform company provides educational resources and customer support to help customers identify the reasons behind failed validations and fix the underlying issues.

## V. Conclusion

Enabling A/B integrations brings new opportunities for A/B testing platforms and for software products that start to offer A/B testing as a feature in their portfolio. Despite many A/B testing platforms available on the market as well as built internally by software companies, integrating A/B testing into products is far from a solved problem. In this paper we discussed our key learnings based on the multi-year journey at Microsoft and Outreach as we integrated A/B testing into products such as Azure PlayFab. We learned that even with the rich infrastructure and processes that we created for 1P A/B testing over the last 15+ years, there is still need to develop and support new scenarios unique to A/B integrations.

Moving forward, we need to dive deeper on the success measured for A/B integrations, and outline the maturity path similar to the Experimentation Growth Model [2] for 1P A/B testing.

TABLE II. Summary of key learnings.

| Learning | Takeaways for A/B platforms | Takeaways for Integrators |
|---|---|---|
| API/SDK infrastructure for A/B integration | • Create a spec that autogenerates code.<br>• Generate events (event-driven arch.)<br>• Support automation and debuggability | • Assign an architect and engineer to create an integration design.<br>• Review and integrate champion scenarios first. |
| Streamline metric editing experience for A/B integrators | • Provide capability for easily shared metrics.<br>• Enable custom metrics | • Standardize logging contract. |
| Enable effective metric computation and analysis | • Use semantic layer for metric and compute definitions. | • Reuse metric definitions for A/B and dashboards. |
| Config service as integrator | • Support integrating with pull and push config.<br>• Enforce A/B trustworthy promises | • Support passing the config separately and jointly with A/B test config. |
| Reusing A/B testing platform UI | • Share design patterns with integrators.<br>• Expose UI component library if possible. | • For PoC and quick ramp-up cross link with the A/B testing platform. |
| Support Education of A/B testing | • Share case studies with leadership and engineering<br>• Provide code examples for trustworthy integrations. | • Add user-facing A/B documentation.<br>• Create courses for users in your domain. |
| Surface validation problems | • Trigger events for failed tests<br>• Expose diagnostic results | • Provide A/A test feature validation.<br>• Expose SRM test result to users. |


## Acknowledgment

We would like to thank everyone at Microsoft, Outreach and Azure PlayFab for contributing to this research and the reviewers that have provided feedback on it.


## References


[1] A. Fabijan, P. Dmitriev, H. H. H. Olsson, and J. Bosch, "The Evolution of Continuous Experimentation in Software Product Development: From Data to a Data-Driven Organization at Scale," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, May 2017, pp. 770–780, doi: 10.1109/ICSE.2017.76.

[2] A. Fabijan, P. Dmitriev, C. McFarland, L. Vermeer, H. Holmström Olsson, and J. Bosch, "Experimentation growth: Evolving trustworthy A/B testing capabilities in online software companies," *J. Softw. Evol. Process*, p. e2113, Nov. 2018, doi: 10.1002/smr.2113.

[3] T. Xia, S. Bhardwaj, P. Dmitriev, and A. Fabijan, "Safe Velocity: A Practical Guide to Software Deployment at Scale using Controlled Rollout," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in*

*Practice (ICSE-SEIP)*, May 2019, pp. 11–20, doi: 10.1109/ICSE-SEIP.2019.00010.

[4] P. Li Luo, P. Dmitriev, M. Hu Huibin, Xiaoyu C., Z. Dimov, B. Paddock, Y. Li, A. Kirshenbaum, I. Niculescu, and Y. Thoresen, "Experimentation in the Operating System: The Windows Experimentation Platform," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, May 2019, pp. 21–30, doi: 10.1109/ICSE-SEIP.2019.00011.

[5] A. Fabijan, H. Holmström Olsson, and J. Bosch, "Differentiating Feature Realization in Software Product Development," in *Product-Focused Software Process Improvement*, 2017, pp. 221–236.

[6] K. Pohl, *Requirements Engineering: Fundamentals, Principles, and Techniques*. 2010.

[7] K. Rodden, H. Hutchinson, and X. Fu, "Measuring the User Experience on a Large Scale : User-Centered Metrics for Web Applications," *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, pp. 2395–2398, 2010, doi: 10.1145/1753326.1753687.

[8] E. Lindgren and J. Münch, "Raising the odds of success: The current state of experimentation in product development," *Inf. Softw. Technol.*, vol. 77, pp. 80–91, 2015, doi: 10.1016/j.infsof.2016.04.008.

[9] T. Barik, R. DeLine, S. Drucker, and D. Fisher, "The bones of the system," in *Proceedings of the 38th International Conference on Software Engineering Companion - ICSE '16*, 2016, pp. 92–101, doi: 10.1145/2889160.2889231.

[10] G. Schermann, J. J. Cito, and P. Leitner, "Continuous Experimentation: Challenges, Implementation Techniques, and Current Research," *IEEE Softw.*, vol. 35, no. 2, pp. 26–31, Mar. 2018, doi: 10.1109/MS.2018.111094748.

[11] E. Ries, *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. 2011.

[12] S. Blank, "Why the lean start-up changes everything," *Harvard Business Review*, vol. 91, no. 5. John Wiley & Sons, p. 288, 2013, doi: 10.1523/JNEUROSCI.0307-10.2010.

[13] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. 2010.

[14] D. G. Feitelson, E. Frachtenberg, and K. L. Beck, "Development and deployment at facebook," *IEEE Internet Comput.*, vol. 17, no. 4, pp. 8–17, 2013, doi: 10.1109/MIC.2013.25.

[15] J. F. Box, "R.A. Fisher and the Design of Experiments, 1922–1926," *Am. Stat.*, vol. 34, no. 1, pp. 1–7, Feb. 1980, doi: 10.1080/00031305.1980.10482701.

[16] S. D. Simon, "Is the randomized clinical trial the gold standard of research?," *J. Androl.*, vol. 22, no. 6, pp. 938–943, Nov. 2001, doi: 10.1002/j.1939-4640.2001.tb03433.x.

[17] A. Fabijan, B. Arai, P. Dmitriev, and L. Vermeer, "It takes a Flywheel to Fly: Kickstarting and Growing the A/B testing Momentum at Scale," 2021, doi: 10.1109/SEAA53835.2021.00023.

[18] A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch, "The Evolution of Continuous Experimentation in Software Product Development," 2017, doi: 10.1109/ICSE.2017.76.

[19] S. Gupta, L. Ulanova, S. Bhardwaj, P. Dmitriev, P. Raff, and A. Fabijan, "The Anatomy of a Large-Scale Experimentation Platform," in *2018 IEEE International Conference on Software Architecture (ICSA)*, Apr. 2018, no. May, pp. 1–109, doi: 10.1109/ICSA.2018.00009.

[20] K. Kevic, B. Murphy, L. Williams, and J. Beckmann, "Characterizing Experimentation in Continuous Deployment: A Case Study on Bing," in *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, May 2017, pp. 123–132, doi: 10.1109/ICSE-SEIP.2017.19.

[21] F. Fagerholm, A. S. Guinea, H. Mäenpää, and J. Münch, "The RIGHT model for Continuous Experimentation," *J. Syst. Softw.*, vol. 0, pp. 1–14, 2015, doi: 10.1016/j.jss.2016.03.034.

[22] R. Kohavi, A. Deng, B. Frasca, R. Longbotham, T. Walker, and Y. Xu, "Trustworthy online controlled experiments," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '12*, 2012, p. 786, doi: 10.1145/2339530.2339653.

[23] R. L. Kaufman, J. Pitchforth, and L. Vermeer, "Democratizing online controlled experiments at Booking. com," *arXiv Prepr. arXiv1710.08217*, pp. 1–7, 2017.

[24] T. Kluck and L. Vermeer, "Leaky Abstraction In Online Experimentation Platforms: A Conceptual Framework To Categorize Common Challenges," Oct. 2017, [Online]. Available: http://arxiv.org/abs/1710.00397.

[25] Y. Xu, N. Chen, A. Fernandez, O. Sinno, and A. Bhasin, "From Infrastructure to Culture," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '15*, 2015, pp. 2227–2236, doi: 10.1145/2783258.2788602.

[26] D. Tang, A. Agarwal, D. O. Brien, M. Meyer, D. O'Brien, and M. Meyer, "Overlapping experiment infrastructure," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '10*, 2010, p. 17, doi: 10.1145/1835804.1835810.

[27] P. Dmitriev and X. Wu, "Measuring Metrics," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management - CIKM '16*, 2016, pp. 429–437, doi: 10.1145/2983323.2983356.

[28] A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch, "The Benefits of Controlled Experimentation at Scale," in *Proceedings of the 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Aug. 2017, pp. 18–26, doi: 10.1109/SEAA.2017.47.

[29] Azure Playfab, "Azure Playfab A/B Testing Documentation." https://learn.microsoft.com/en-us/gaming/playfab/features/analytics/ab-testing/.

[30] A. B. Sandberg, "Agile Collaborative Collaboration," *IEEE Comput. Soc.*, vol. 28, no. 4, pp. 74–84, 2011, doi: 10.1109/MS.2011.49.

[31] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empir. Softw. Eng.*, vol. 14, no. 2, pp. 131–164, 2008, doi: 10.1007/s10664-008-9102-8.

[32] A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch, "Effective

online controlled experiment analysis at large scale," in *Proceedings - 44th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2018*, 2018, pp. 64–67, doi: 10.1109/SEAA.2018.00020.

[33] A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch, "Online controlled experimentation at scale: An empirical survey on the current state of A/B testing," in *Proceedings - 44th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2018*, 2018, pp. 68–72, doi: 10.1109/SEAA.2018.00021.

[34] A. Fabijan, "Diagnosing Sample Ratio Mismatch in A/B Testing." https://www.microsoft.com/en-us/research/group/experimentation-platform-exp/articles/diagnosing-sample-ratio-mismatch-in-a-b-testing/.

[35] J. Doerr, *Measure what matters: How Google, Bono, and the Gates Foundation rock the world with OKRs*. Penguin, 2018.

[36] W. Machmouchi and G. Buscher, "Principles for the Design of Online A/B Metrics," in *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval - SIGIR '16*, 2016, pp. 589–590, doi: 10.1145/2911451.2926731.

[37] P. Dmitriev, B. Frasca, S. Gupta, R. Kohavi, and G. Vaz, "Pitfalls of long-term online controlled experiments," in *2016 IEEE International Conference on Big Data (Big Data)*, Dec. 2016, pp. 1367–1376, doi: 10.1109/BigData.2016.7840744.

[38] P. Dmitriev, S. Gupta, K. Dong Woo, and G. Vaz, "A Dirty Dozen: Twelve Common Metric Interpretation Pitfalls in Online Controlled Experiments," 2017.

[39] "Airbnb-engineering: how-airbnb-achieved-metric-consistency-at-scale." https://medium.com/airbnb-engineering/how-airbnb-achieved-metric-consistency-at-scale-f23cc53dea70.

[40] S. Patotski, "Metric-computation-for-multiple-backends." https://www.microsoft.com/en-us/research/group/experimentation-platform-exp/articles/metric-computation-for-multiple-backends.

[41] "Airbnb-engineering: how-airbnb-achieved-metric-consistency-at-scale." .

[42] S. Patotski, "Metric-computation-for-multiple-backends." . https://www.microsoft.com/en-us/research/group/experimentation-platform-exp/articles/metric-computation-for-multiple-backends

[43] M. T. Rahman, L.-P. Querel, P. C. Rigby, and B. Adams, "Feature Toggles: Practitioner Practices and a Case Study," *Proc. 13th Int. Work. Min. Softw. Repos. - MSR '16*, pp. 201–211, 2016, doi: 10.1145/2901739.2901745.

[44] W. Vogels, "Continuous Configuration at the Speed of Sound | All Things Distributed," *All Things Distributed - Amazon*, 2021. https://www.allthingsdistributed.com/2021/08/continuous-configuration-on-aws.html.

[45] N. Chen, M. Liu, and Y. Xu, "Automatic Detection and Diagnosis of Biased Online Experiments," Jul. 2018, [Online]. Available: http://arxiv.org/abs/1808.00114.

[46] A. Fabijan *et al.*, "Diagnosing Sample Ratio Mismatch in Online Controlled Experiments," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining - KDD '19*, 2019, pp. 2156–2164, doi: 10.1145/3292500.3330722.

[47] A. Fabijan *et al.*, "Diagnosing sample ratio mismatch in online controlled experiments: A taxonomy and rules of thumb for practitioners," 2019, doi: 10.1145/3292500.3330722.