Safe Velocity: A Practical Guide to Software Deployment at Scale using Controlled Rollout

Tong Xia
Microsoft Corporation
Redmond, WA, USA
toxia@
microsoft.com

Sumit Bhardwaj
Microsoft Corporation
Redmond, WA, USA
subhardw@
microsoft.com

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

Aleksander Fabijan
Microsoft Corporation,
Redmond, WA, USA
aleksander.fabijan@
microsoft.com

***Abstract*—Software companies are increasingly adopting novel approaches to ensure their products perform correctly, succeed in improving user experience and assure quality. Two approaches that have significantly impacted product development are controlled experiments – concurrent experiments with different variations of the same product, and phased rollouts - deployments to smaller audiences (rings) before deploying broadly. Although powerful in isolation, product teams experience most benefits when the two approaches are integrated. Intuitively, combining them may seem trivial. However, in practice and at a large scale, this is difficult. For example, it requires careful data analysis to correctly handle exposed populations, determine the duration of exposure, and identify the differences between the populations. All of these are needed to optimize the likelihood of successful deployments, maximize learnings, and minimize potential harm to users of the products. In this paper, based on case study research at Microsoft, we introduce controlled rollout (CRL), which applies controlled experimentation to each ring of a traditional phased rollout. We describe its implementation on several products used by hundreds of millions of users along with the complexities encountered and overcome. In particular, we explain strategies for selecting the length of the rollout period and metrics of focus, and defining the pass criterion for each of the rings. Finally, we evaluate the effectiveness of CRL by examining hundreds of controlled rollouts at Microsoft Office. With our work, we hope to help other companies in optimizing their software deployment practices.**

Keywords—Controlled rollout, controlled experiment, software development, phased rollout

I. INTRODUCTION

In late November 2017, Wired magazine reported that Mozilla will be releasing Firefox Quantum and that based on the reporter's evaluation, it was the browser to use. From a software engineering perspective, the interesting part here is how Firefox was released. Long before this press article, a subset of users was using pre-release versions of Firefox providing telemetry to Mozilla which was using this data and the users' verbatim feedback as the software was being developed. Mozilla Corporation – the company building Firefox – is not an exception here. Almost every software – open-source or proprietary – has an early-access program usually called the "beta" or "insider" program that allows a subset of users to see the upcoming features and provide feedback to the developers. For example, Microsoft Windows Insiders [1] has over 1 million users. This approach of gradually rolling out a new version of software rather than making the release available to all users instantly is known as *phased rollout* [2]. The value of phased rollout comes from providing a feedback mechanism from the users to the developers, attempting to ensure that the software meets quality standards and satisfies users' needs.

However, as we discuss below, phased rollouts suffer from several problems: they often take a long time, the early audience is not representative of the overall population, it is difficult to accurately measure the impact of the changes, and as a result the feedback and the input into the next iteration of the product are limited.

A widely used tool for accurately evaluating the impact of a new feature in software is Online Controlled Experiment (OCE), or A/B test [3], [4]. In an OCE, the new feature is made available for a randomly sampled group of users of the product – also known as treatment, and the impact of the feature on the quality of the software is compared to that of an equivalent group of users who have a different version of the feature or do not have the feature – also known as control. If the difference (or delta) between treatment and control is statistically significant, then with high probability the change (i.e., the new feature) caused the observed difference. Establishing this causal relationship between the change made to the product and the difference in the quality of software is something that the phased rollout does not provide.

OCEs however, are inherently riskier, requiring a true random sample of production users to obtain trustworthy results. They also do not produce as much useful verbatim feedback, compared to the early stages of phased rollout [5], [6].

In this work, we introduce a hybrid approach that we call *controlled rollout* (CRL), which integrates OCE into phased rollout process and addresses some of the problems mentioned above. Controlled rollout works across different software development methodologies, such as agile, waterfall, hybrid, and can be used by software development teams of any size.

While the concept of controlled rollout is simple, its implementation in practice is non-trivial and requires careful data analysis and engineering skills to answer several important questions to ensure optimal and correct usage. In this paper, we discuss these questions and the framework to derive the optimal answers based on our experience of implementing controlled rollouts at Microsoft. We use Office client applications as an example to illustrate how we apply controlled rollouts to streamline and inform release management in products.

Although the difference between phased rollout and controlled rollout appears to be small, the implications are

profound – both in terms of the value provided and the impact on software development and deployment. For example, controlled rollout enables measurement of the impact of an individual feature in isolation which is not possible with phased rollout. As a result, the product team can formulate hypotheses on whether their change will lead to impact in the quality of software and if the hypothesis is not borne out in data, and alter the subsequent decisions on software development. One can think of controlled rollout providing a closed loop for software development with the user in the middle.

The key contributions of this paper are as follows:

- We discuss the challenges in modern release deployment management, and why neither phased rollout nor controlled experiments in isolation is adequate;
- We propose controlled rollout, a hybrid approach that combines the strength of controlled experiments and those of phased rollout;
- We describe the challenges of implementing and correctly applying controlled rollout, the data analysis that needs to be performed to do it correctly, illustrate the discussion with real examples from Microsoft, and evaluate the impact on a dataset of hundreds of real controlled rollouts that ran at Microsoft;
- We discuss practical challenges of introducing controlled rollout into large mature software products.

The remainder of the paper is organized as follows. We first provide a background on several related topics in Section II and discuss our research method in Section III. We introduce controlled rollout and discuss implementation choices and guidelines in Section IV. Sections V discusses challenges of introducing controlled rollout into established engineering processes. We conclude and list directions for future work in Section VI.

II. BACKGROUND

A. Software Deployment

Software deployment is the process of releasing a new software *build* (a set of new functionalities, updates, and bug fixes) to the users [2]. Depending on the type of software and the software engineering process used by the development team, the size (amount of changes in the build) and frequency of deployments vary. Organizations that have adopted agile software development [7], especially continuous delivery and deployment [2], [8], generally see higher frequencies of releases which are also smaller in size, compared to those following traditional software development methodologies such as waterfall.

Due to the distributed nature of development and deployment of modern software, the variety of operating scenarios such as devices, OS versions, network types, increasing use of artificial intelligence (A.I.) that learns from users and updates over time, and the fact that software is frequently connected to various services which are deployed independently, it is extremely challenging, perhaps impossible, and expensive to comprehensively test and ensure software quality via internal testing or in the lab [9]. Complexity of modern software also makes it increasingly difficult to control technical debt [10] (removing or refactoring old code) while ensuring there is no degradation in quality. Deploying the software to users and customers is required to obtain good validation coverage of the scenarios mentioned above [11]. To obtain such exposure and ensure the quality of builds, both agile and traditional software engineering companies are employing the process of phased rollout, where the new build is deployed in several steps or phases [12].

Phased rollout proceeds as follows. The deployment of the build starts with a small group of users, the build's quality and the value of the product is validated within that group, after which it is deployed to a larger group, and so on until the build reaches all users. We will refer to these groups of users as *rings*. Fig. 1 shows a possible split of all users of a product into a sequence of rings. Typically, the first one or two rings would be internal, including only company employees, followed by 1-3 external rings, before reaching the production ring that includes all users. Naturally, users in the initial rings will be exposed to the highest number of build releases, many of which may be rolled back due to issues, while users in the later rings will receive less frequent, more stable builds. For example, Microsoft Windows uses two internal rings and allows external users to opt into one of the three external rings: fast, slow, and release preview [1]. Google Chrome has four rings (aka channels): canary, dev, beta, and stable [13]. Many other products and companies follow a similar approach.

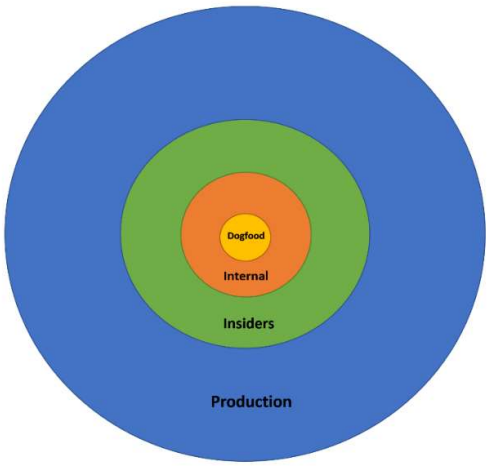

Fig. 1. Users of a product are split into rings.

The selection of users into rings is typically an opt-in process, attracting early adopters [14], bloggers, system and website administrators, as well as developers who want to take early advantage of new features. The validation of build quality inside a ring is done using two main approaches:

1. **Verbatim user feedback:** Users who opted into rings are typically more proactive and have more interest in ensuring the quality of the software than the general audience, resulting in higher willingness to provide verbatim feedback as well as ability to provide more precise and actionable feedback.
2. **Monitoring of quality metrics:** Key quality metrics, such as crash rate and load time, are monitored. An increase in such a metric after the deployment of a new build indicates a quality issue with the build.

This process of ensuring quality of builds, while widely used, has several problems. First, the process is slow. The

build needs to stay in a ring for a considerable period to both allow users the time to use the feature and provide enough feedback, and to collect enough data to ensure there are no quality regressions in metrics that are monitored which may be highly variable due to the day of week, holiday, and other external factors. There is no natural notion of the "optimal" time to keep the build in each ring, resulting in ad-hoc and highly variable procedures used in practice. These issues slow down the process, increasing the time to detect an issue and the time it takes for a good build to make it to production.

Second, only sufficiently large quality issues can be detected during the phased rollout process. Users will only provide feedback on the features they notice or experience, and, due to high variability of metric values over time, metric monitoring process typically can only reliably detect large changes in metrics, e.g. 10% or larger. Issues affecting a segment of the population not well represented in a specific ring are likely to go unnoticed as well [15].

Third, the process is mostly concerned with the reliability aspect of build quality, such as crashes and performance, rather than with the impact of the build on user experience and the ability of users to successfully and in a timely manner achieve their objective. But, as described in [16], most new features do not have the intended positive effect on users that the developers hoped for, leading to increased desire to measure the impact on the success of users to complete their objectives in addition to ensuring the build's quality. Learning about the impact of the new build on user experience as early as possible is also important for estimating the value of feature for the customers, as well as for informing the design of the next iteration of the product. While verbatim feedback often contains qualitative information about how much users like or dislike new features [5], this feedback is highly skewed - users who disliked the feature are generally more likely to provide feedback than those who liked it – and does not provide an objective way to evaluate the overall impact. Also, since in established products new releases have only slight incremental impact on user experience, the change in user experience is often too small to be detected.

The issues mentioned above serve as a motivation for us to combine phased rollout with controlled experiments - a proven mechanism for evaluating the impact of changes.

B. *Online Controlled Experiments*

Online Controlled Experiments, or A/B tests, are widely used by data-driven companies to evaluate the impact of new features on user experience in websites [17], [18], mobile and desktop apps, gaming consoles, social networks [19], and operating systems [20]. In the simplest controlled experiment, users are randomly assigned to one of the two variants: control (A) or treatment (B). Usually control is the existing system and treatment is the system with a new feature added, say, feature X. User interactions with the system are recorded and from that, metrics are computed. If the experiment was designed and executed correctly, the only thing consistently different between the two variants is the feature X. External factors such as seasonality, impact of other feature launches, moves by competition, etc. are distributed evenly between control and treatment and therefore do not impact the results of the experiment. Hence, any difference in metrics between the two groups can be attributed to either the feature X or noise. The noise hypothesis is ruled out using statistical tests such as t-test [21]. This establishes a causal relationship between the change made to the product and the change in user behavior, which is the key reason for widespread use of controlled experiments [4].

Controlled experiments address the three problems with ensuring the software quality discussed above [9]. A power calculation can be done to determine the optimal length of time to run an experiment to detect the impact we are interested in, which, because of the increased sensitivity in detecting changes, is usually much shorter than the full rollout process. Due to all external factors being randomized between control and treatment, very small changes in metrics (e.g. less than 1%) can often be reliably detected. Finally, rather than focusing on what users say they do, like, or dislike, experiment analysis focuses, via a rich set of metrics, on what users actually do, resulting in a more precise and unbiased evaluation of the new feature.

The above advantages of controlled experiments resulted in more and more companies starting to evaluate changes to their products with experiments, in addition to validating build quality with phased rollout [22]. The two techniques are generally used complementarily. First, a build containing a number of inactive, or dark, features is rolled out to production. Then, experiments are run on these features to determine their impact. The winning features are then activated for all users. In subsequent builds, the experimentation configurations are removed to make these features the default experience, reducing the technical debt introduced during the process [23]. At Microsoft, many products employ both controlled experiments and phased rollout.

The problem with the above approach, however, is increased risk. To obtain unbiased results, experiments need to be run on the production population. But launching a new feature to production always carries a risk of an undetected bug causing a crash, hang, or other severe degradation. Another problem is that less verbatim feedback is obtained during the evaluation. Only a randomly sampled subset of the inner ring users would see the feature during the evaluation and be able to submit the feedback. Finally, the above approach still suffers from the slowness of the traditional phased rollout process that the new build needs to go through before being evaluated via an experiment.

In section IV, we describe a solution that combines the benefits of phased rollout and controlled experiments, while addressing the above problems.

III. RESEARCH METHOD

The research presented in this paper has been induced through a case study at Microsoft Corporation in the USA between July 2017 and August 2018. The authors of this paper are or have been employed at the Analysis and Experimentation (A&E) team of Microsoft. The A&E team provides a platform and services for running controlled experiments and rollouts. Its data scientists, engineers and program managers are involved with product teams and departments across Microsoft every day. In a year, over ten thousand trustworthy controlled experiments and rollouts are conducted and analyzed.

In the remainder of this section, we briefly describe our case study based on the recommendations described by Runeson and Höst [24].

Data Collection. The primary sources of data were the following. First, we used historical data such as product logs, experiment scorecards, experiment metadata, and meeting minutes. Second, we collected internal documentation such as documents describing the experimentation process, platform, and the aforementioned phased rollout approach. Third, we collected notes and screenshots of real experiments that ran at MS Office (see e.g. Fig. 3). Finally, the first three authors of this paper had daily hands-on experience in running experiments and access to workshops, meetings and tutorials with other practitioners at Microsoft.

Data Analysis. We employed both qualitative and quantitative approaches to data analysis. First, the authors of this paper jointly analyzed the collected data through thematic coding [25] to identify, for example, the common characteristics of a ring such as its size and average length of the experiments in that ring. This analysis was particularly useful to empirically induce a description of our main contribution – the CRL approach. On the other hand, to evaluate our proposed approach, we analyzed several hundreds of experiments that ran at MS Office through descriptive statistics. In particular, we examined how many experiments in a CRL provided insightful results to product teams for a quicker reaction when compared to releases that were done through the traditional phased rollout. We provide more details about this part of the analysis in Section V.

Threats to Validity. Runeson and Höst [24] recommend the use of different kinds of triangulation in order to mitigate several kinds of threats to validity in case study research. We applied several of these triangulation techniques, in particular, *using confirming evidence from multiple sources* (getting the same answers from data scientists, engineers, and program managers), *using different methodologies* (employing quantitative and qualitative methods), and *having several data collectors*. Despite these efforts, however, our research risks several threats to validity [24].

Construct validity. To mitigate construct validity threats, we conducted the research in a setting where everyone was very well familiar with the terminology and elements of experimentation, phased rollout etc. through prolonged involvement [26].

External validity. The theory induced in this research is applicable and most valuable for product organizations and companies that share common characteristics with our case company. In particular, these are the companies that are able to release software in phases to different audiences and have the knowhow and infrastructure to conduct online controlled experiments.

Reliability. To mitigate reliability threats, we employed member-checking and peer-debrief techniques [26]. In particular, multiple researchers reviewed the data as well as the induced theory, and over a dozen of practitioners working at the case company that were not directly involved in this study provided feedback on our work.

IV. PRACTICAL GUIDE TO CONTROLLED ROLLOUT

In this section, we introduce the approach of Controlled Rollout (CRL). In particular, we discuss the strategies for designing and organizing rollout metrics, for deciding on rollout durations, and strategies for designing and decision making in individual rings. We illustrate our approach and the individual contributions with a practical example.

A. Controlled Rollout Essentials

In a CRL, the deployment follows the phased rollout process described in the previous sections. The main difference is that during each phase of the rollout (in each ring) a controlled experiment is run. The reliability and impact of the build on the users in the ring is measured, and, if pre-specified criteria are met, the rollout proceeds to the next ring starting a new experiment there, while fully deploying the build to the preceding ring.

While the idea of CRL is simple, its implementation in practice poses many challenges: the audience within a specific ring is typically not representative for all production users, different users may be installing the build and getting into the experiment at different points in time creating further bias towards early adopters, the number of users in early rings may be small, the criteria for advancing to the next phase need to be defined so as to balance the reliability of the build, the learnings about user impact, and the speed of moving to the next phase.

Since each ring has different type and number of users, the types of metrics and statistical criteria for advancing to the next phase should also differ from ring to ring. The larger the number of users in the experiment, the smaller the magnitude of changes in metrics that the experiment is able to detect [27]. Therefore, in general, in the early and small rings, one should focus on larger effects, such as harmful degradations in software quality. In later and larger rings, one should shift the focus to smaller effects, positive or negative, on user engagement and satisfaction. Table 1 summarizes our recommendations for the basic ring setup described in Fig. 1 above, which we elaborate in more details below.

B. Rollout Metrics

A rich set of metrics is required for different phases of a controlled rollout. In this subsection, we review the major types of metrics that are created for most controlled rollouts at Microsoft. We differentiate between *global metrics* (e.g. Data Quality, Guardrail and Success metrics), which are applicable to most controlled rollouts, and *local metrics,* which are applicable to individual rollouts. We hope that the structure below can be used as a practical guide for metric design and interpretation of results of controlled rollouts.

Data quality metrics. These metrics are created to ensure that each phase of a controlled rollout is set up correctly and that the results of the experiment are trustworthy. Statistically significant difference on any of these metrics in any direction indicates an issue. One of the most important data quality metrics is the ratio of the number of users in each group, or sample ratio. A Sample Ratio Mismatch (SRM) happens when the observed ratio is different from expected, indicating that either the controlled rollout was not set up correctly or there is some missing data / redundancy issue that impacts the treatment and the control groups unevenly.

TABLE 1. Rings of an example of controlled rollout.

Ring Name	Size (user count)	Length of Rollout Phase	Focused Metrics	Pass Criterion
Dogfood	Very small. From less than one hundred to several hundreds.	From days to 1 week.	Guardrail metrics, Local metrics	Feature is being triggered and users are engaging with it. No serious bug or crash that moves guardrail metrics by over ~10%.
Internal	Small. Hundreds to thousands.	Up to 1 week.	Guardrail metrics	No serious bug or crash that moves guardrail metrics by over ~10%. No negative impact on product usage.
Insiders	Medium to large. Tens to hundreds of thousands.	2-4 weeks.	OEC metrics, Local metrics, Guardrail metrics	Positive movement on local metrics that measure the direct impact. No negative impact on guardrail, OEC or other local metrics.
Production	Very large. Hundreds of thousands to millions.	2-4 weeks or longer.	OEC metrics Local metrics Guardrail metrics	Effectiveness of the new feature, which is reflected by improvement on OEC or local metrics. May require a trade-off between gains and losses among different metrics.

SRM violates the basic principle of controlled experiments that the treatment and the control groups must be selected randomly, and typically completely invalidates the results. Other common data quality metrics measure the rate of data delay and loss, reliability of instrumentation, cookie churn, etc. The first step in analyzing any phase of a controlled rollout is to ensure that data quality metrics did not move.

Guardrail metrics. These metrics represent important business constraints that should not degrade as a result of the rollout. Examples of guardrail metrics are application crash and hang rate, service load time, number of page views, etc. One of the major goals of phased and controlled rollout is to ensure that guardrail metrics do not degrade with the deployment of a new build.

Success metrics. Also referred to as Overall Evaluation Criteria (OEC) [28]. These are the key metrics which indicate that the new build or feature are likely to lead to an increase in long-term business goals, such as user engagement and retention. Good success metrics are often difficult to define, as many obvious choices do not work. For example, short-term revenue is generally not a good choice because it is easy to increase it by, e.g., showing users more ads, which may result in user dissatisfaction and abandonment over the long term. Accurately estimating the long-term impact involves many factors and is challenging [29], [30].

Local metrics. These are metrics that directly measure users' engagement with the new features being tested. While data quality, guardrail, and success metrics apply equally to all controlled rollouts and therefore are often referred to as "global metrics", local metrics are of interest specific to the rollout and the new feature. One of the most important types of local metrics is "*feature usage metrics*". These metrics track whether the new feature is triggered correctly and measure how frequently users interact with it. For example, if a new menu item was added to the product, the number of clicks on that item could be a good feature usage metric. In many cases, feature usage metrics do not apply to the control group which was not exposed to the new feature. Feature owners monitor these metrics to check if the feature is triggering and working as expected. Besides, there should also be local metrics that measure the impact of the new feature at a higher level, and these metrics should be applied to both the treatment and control groups. Following the menu item example, suppose the new item is an extra way of sharing a Word document with other users. Feature owners should consider creating metrics that measure document share activities and users' engagement with the menu as a whole. Such metrics are an important indication of whether users' needs to complete their objectives are met. They show the level of engagement with the new features relative to other existing features, and help to understand deeper and debug the movements observed in global, guardrail and success metrics.

Local metrics are usually closer to specific feature changes, and because of that many learnings for future iterations of the product come from understanding which local metric changes lead to improvements in global metrics. While phased rollout can capture some local metric changes, controlled rollout is able to measure more changes more accurately due to much higher sensitivity.

C. Length of Rollout Periods

Another key question is how long the controlled rollout should run at each ring. This is a trickier problem than it seems. On one hand, the running period needs to be long enough so that feature owners can collect the data they need. On the other hand, it cannot be too long since feature owners want to iterate quickly in the current ring and move forward to the next ring. The optimal solution should align with the objectives of the ring and the availability of data.

In this subsection we describe a practical method to determine the length of the experiment period based on the theory of hypothesis tests. In the next subsection we will discuss how to combine this theory and the objectives of the ring to choose the proper rollout period for each ring.

In a typical controlled experiment, a two-sample t-test is applied to each metric to determine whether its values in the treatment and the control groups are significantly different from each other. The t-test constructs the test statistic by dividing the difference on the mean value of the two samples by the combined sample standard deviation. This test statistic follows a t-distribution with degree of freedom equaling to the total sample size minus two. When the sample sizes of the experiment groups are large, which is the case for the majority of online products, the distribution of the test

statistic approximates to the standard normal distribution. This distribution will be used for making the final inference.

Let 1-α be the confidence level, β be the statistical power and $z(\cdot)$ be the inverse cumulative distribution function of the standard normal distribution. For simplicity, assume the treatment and the control groups have the same number of users. It can be shown that, in order to be able to detect a Δ% change of a metric at confidence level 1-α and power β, we need at least n users in each group, where

$$n = \left(\frac{s\left(z\left(\frac{\alpha}{2}\right) + z(\beta)\right)}{\mu \Delta\%} \right)^2 \quad (1)$$

Here μ is the mean value of the metric in the control group, and s is the standard deviation of the metric. In practice, α and β are set to be 0.05 and 0.8 respectively for most cases. μ and s can be estimated from the data.

Another option is to use the approximation formula as described in [31].

$$n = \frac{16\sigma^2}{(\Delta\%)^2 \mu^2} \quad (2)$$

Feature owners need to set the proper target of Δ% based on the size of the ring and the objective. It can be seen from (1) or (2) that the required number of users n is inversely proportional to the square of the sensitivity Δ%. In the early rings "Dogfood" and "Internal", where the set of users is usually small, Δ% has to be relatively large so that the user count n that makes the equation hold can be achieved within a reasonable period of time. When the controlled rollout enters later rings, the available number of users becomes much larger. Hence feature owners can set up a more aggressive target on Δ%.

Once feature owners have a target on Δ%, they can substitute it together with other parameters into Formula (1) or (2) and get the number of users needed to achieve the goal. They should then choose an experimental period which is long enough to collect this number of users.

If feature owners plan to run controlled rollouts on a new product, they should run an A/A test on each of the rings first. An A/A test is a trial experiment where the treatment and control groups are exposed to exactly the same experience. This test ensures that the experiment configuration works correctly in the new setting, and it also help feature owners learn about the data. How many users do they collect in a given period of time? What are the metric values μ and standard deviation s? With this learning, feature owners can set a realistic target on the sensitivity Δ%, calculate the required user count n from Formula (1) or (2), and decide on the length of the controlled rollout.

D. Strategies for Each Ring

The controlled rollout is conducted through different rings each having a different number and type of users. Therefore, feature owners need to define the goal, choose the exposure duration, select the set of metrics, and determine the pass criterion separately for each ring.

Dogfood. This group includes developers of the product and internal employees who volunteered to test the new features. It is the earliest ring of controlled rollout. At this beginning stage, the first goal is to verify that the new feature is triggered correctly. Feature owners should check the feature usage metrics and make sure the values are expected. Quality of instrumentation and correctness of new metrics should also be verified using local metrics. At this early stage, new features are usually not fully mature, and have a relatively high possibility to behave unexpectedly. Therefore, the second goal in this ring should be trying to identify egregious events, such as crashes and bugs, within a short period of time. Feature owners should check for dramatic (10+%) negative movement on guardrail metrics. If such movement is observed and identified as statistically significant, they should shut down the current rollout as soon as possible to stop hurting users. Then they will need to debug their feature based on the guidance from the metrics and quickly start a new iteration. Normally, controlled rollouts will stay in this ring for several iterations before all bugs are fixed. When no significantly negative movement on guardrail metrics is observed, feature owners should promptly move forward to the next ring.

Fig. 2 plots the detectable change on a representative crash metric against the length of the rollout period for Office 365 in the "Dogfood" ring.

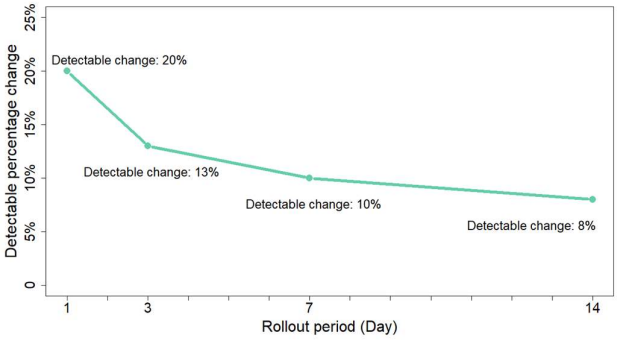

Fig. 2. User count and detectable change vs rollout duration.

The values are estimated using the method described in the previous subsection. For our example, it can be seen that the rollout needs to be run for at least 7 days in order to detect a 10% change in this metric. On the other hand, increasing the duration from 7-day to 14-day, we only see ~20% more new users in the rollout, which improves the detectable percentage change to about 8%. This improvement, in general, makes only a small difference in the results. Considering both the improvement in metric sensitivity and the time cost, 7-day is the optimal duration.

Internal. This ring includes internal employees who use the product regularly. The size of this ring is slightly larger than the previous one, ranging from hundreds to thousands of users, depending on the organization. The objective of controlled rollout in this ring is similar to the "Dogfood" ring. Feature owners should still focus on checking for crashes and bugs. But since most of the egregious events have been cleared off in the "Dogfood" ring, feature owners can focus on less sensitive guardrail metrics.

At the early rings "Dogfood" and "Internal", compared to the traditional phased rollout technique, controlled rollouts

are more sensitive in detecting the impact of new features. If there is a seriously harmful effect, a controlled rollout can usually identify it in days as double-digit percentage change on its guardrail metrics. At Microsoft, we have a monitoring and alerting pipeline coupled with our rollout system. Data scientists and feature owners will be notified immediately when there is a significant movement on a metric that exceeds the pre-defined threshold. They can even require the system to automatically shut down the rollout if the effect is "extremely harmful", which is defined as dramatic negative movement on some key metrics. This mechanism allows feature owners to quickly iterate on their new features. On the other hand, if there is no egregious event, the controlled rollout is able to confirm this within 7 days and allow feature owners to move to the next ring. In contrast, in a phased rollout, feature owners may have to wait for weeks to gather enough feedback from dogfood users and paid testers, then analyze it and decide whether their feature is safe to be promoted to the next ring.

Insiders. This ring includes external users who opted in to try new features of a product before they are officially released. In this ring, there is a much larger set of users, normally hundreds of times larger than the earlier rings. This enables feature owners to detect much smaller percentage change on most metrics. Therefore, their objectives should be adjusted accordingly. The first goal is to still check for guardrail metrics. Although the most harmful bugs were already identified and fixed, it's not uncommon to find single-digit percentage point degradations in guardrail metrics which were not detected in the earlier rings. In addition, the behavior of users in this ring can be very different from the previous two rings. Users in "Dogfood" or "Internal" use the product in a somewhat homogeneous way, since they are from the same organization and work on similar tasks. In contrast, users in the "Insiders" ring tend to be more diverse and are likely to trigger new bugs. The second goal is to start monitoring OEC and local metrics. Local metrics which measure the direct impact of the new feature should show significant positive movement in this ring. Otherwise, it may indicate that the new feature is not being noticed by the users or may not be functioning correctly. Also, OEC metrics should be at least flat. Any significantly negative movement on OEC metrics will be an alarming sign. When this happens, feature owners need to dive into the data to understand the reason before moving to the next ring. In [15], we described an effective approach to analyze such cases.

At Microsoft, based on the data that we collect in the "Insiders" ring, the detectable percentage change Δ% is around 3% for guardrail and some of the local metrics, and around 1% for OEC metrics. Using the estimation method discussed in the previous subsection, we found that most controlled rollouts should run for 2 weeks. This time period is not just for collecting enough data points to meet the goal on Δ%, but is also the minimum length required by some OEC and local metrics. For example, the "retention rate" metric, by definition, only counts users whose activities cross a given time span, such as 7 days. Typically, the controlled rollout needs to run for at least twice as long for these metrics to be valid and gather enough users.

In the traditional phased rollout process, one of the most important ways to evaluate the new feature at the Insiders ring is to analyze user feedback. Although it directly reflects the impression to the new feature of some users, user feedback does not provide a full picture. For example, users only provide feedback on what they see, but hardly noticeable features can sometimes have huge business impact. An example of such feature is the tuning of the font colors of Bing.com search result pages in 2013 (page 3 of [32]). The change was so small that almost no one could tell the difference until seeing both treatment and control colors side by side. However, that change generated an additional ~10 million dollars annually for Bing. User feedback alone cannot provide a comprehensive evaluation on efficiency, user retention, performance cost and crashes, which controlled rollouts do. Feedback can also be biased. Users who leave feedback generally engage with the product more actively than users who do not. If product owners just look at user feedback, the impact of the product on less active users may be ignored. Controlled rollout complements user feedback with its rich set of well-designed metrics calculated from tens or even hundreds of thousands of users.

We illustrate the last point in the previous paragraph with an example of a controlled rollout which ran at Microsoft in the first half of 2017. The Word product team in the Office group conducted a controlled rollout on a feature that changed the default view of Word on Android devices from the "print layout view" to the "mobile view" (Fig. 3), which allocated more screen space to the document, and zoomed in to make it easier to edit.

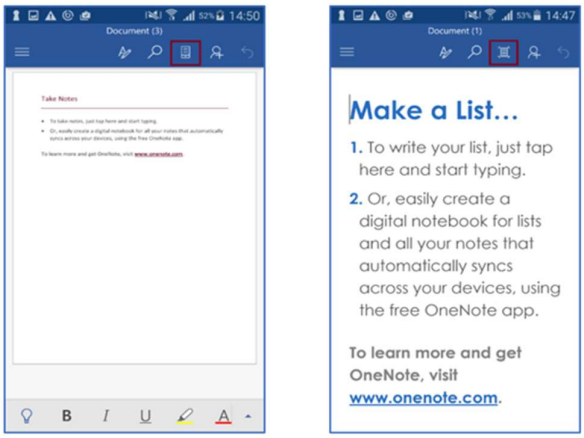

Fig. 3. The print layout view (control, left) and the mobile view (treatment, right) for Word on Android.

When the rollout was executed in the Insiders ring, feature owners also collected user feedback. Most users said they really liked this new view because they could easily edit the document. However, the experiment results were mixed. The engagement metrics moved positively only for users who had modified at least one document (through either create or edit action) during the experiment period. This group of users was more active than the other group, but they only accounted for about 30% of the whole user set. For the remaining group, which contained users who just opened and read documents, the engagement metrics actually moved in the opposite direction. In addition, the feature owners observed a 13% increase in "quick view switches", which is a local metric that measures how often users switched to the non-default view

within 5 seconds after the document was opened. This result suggested that setting the mobile view as default was not well accepted by all users. As a result, the Word team decided to redesign the feature. They eventually rolled out a less aggressive version, which showed the mobile view by default only on those documents which users opened before and closed while in the mobile view.

Production. This ring includes the general group of external users of the product. When the controlled rollout completes at this last ring, feature owners need to decide whether they should officially release their feature, and they should mainly rely on OEC metrics to make this decision. In general, in order to ship a feature, there need to be significant positive movements on OEC metrics. Some controlled rollouts, however, are for minor feature changes which themselves may not have significant impact on OEC metrics, in which case local metrics can be used, provided that OEC and guardrail metrics do not degrade. The "Production" ring is the largest of all the rings of controlled rollouts. At Microsoft, most products have millions to hundreds of millions of users in this ring. Such a large user set allows us to set very aggressive targets on the detectable percentage change Δ%, usually at around 0.5% or smaller. This is necessary at least for the OEC metrics, which, in general, are hard to move [32]. In order to detect such small change, controlled rollouts need to run for at least 2-4 weeks.

In contrast, the traditional phased rollout process evaluates the impact of the new feature by comparing the pre-period data to that in the post-period (after the release of the feature). This analysis can easily be biased by many factors, such as periodical / seasonal effects or the interference of other features that were rolled out during the same period. The traditional method can only reliably detect changes of 10% or higher, the magnitude rarely observed in OEC metrics.

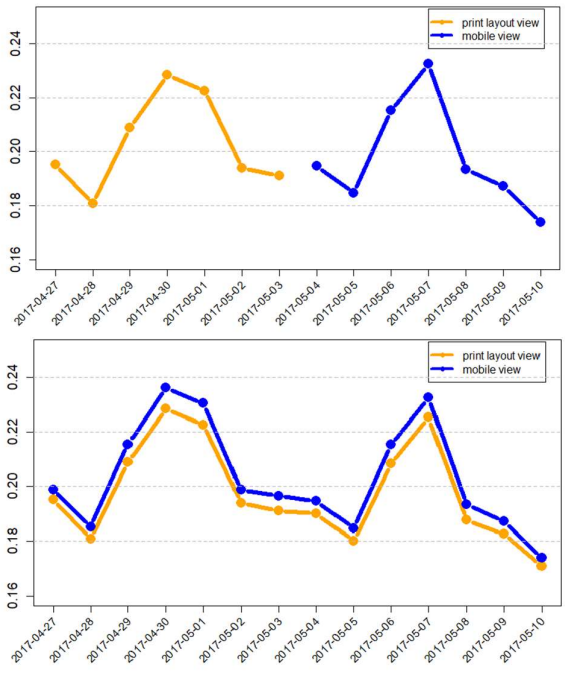

Fig. 4. Metric values on two variants in the cases of phased rollout (top) and controlled rollout (bottom).

Fig. 4 plots the values of one of the OEC metrics of the "mobile view" rollout described above. The top part shows what the metric looks like if we keep the current "print layout view" in the first week, and roll the new "mobile view" out in the following week. It can be seen that the metric values change from day to day, and it is very difficult to decide which view has a higher value on this metric. In contrast, the bottom part shows the values of this metric in the controlled rollout. In this case, it is much more obvious that the metric value for the "print layout view" (control group) is consistently lower than the value for the "mobile view" (treatment group). From hypothesis test, it can be shown that on average, the metric is about 2.5% higher for the "mobile view" when compared to the "print layout view". The p-value is less than 10^{-25}, which indicates that the difference is very significant. This example shows that controlled rollouts are able to detect much smaller percentage changes. As a result, it allows feature owners to detect changes in a much wider range of metrics in a much more detailed way.

V. Evaluation of CRL

In this section, we evaluate the effectiveness of controlled rollout technique by examining the logs of controlled rollouts on Microsoft Office client apps during a several months long period of time in 2018.

MS Office. At Microsoft Office, there are hundreds of controlled rollouts in each ring. For each rollout, there are scorecards [15] that contain the corresponding metrics for periods of the first 3 days, first 7 days and longer time ranges (14, 21 or 28 days if the rollout ran for that long).

In reviewing the logs, and to evaluate the effectiveness of controlled rollouts, we try to answer the following questions: **(1) How often do controlled rollouts "outperform" the traditional phased rollout process**? in terms of, e.g., detected issues, moved to the next ring faster, or provided more insights, and **(2) was the length of rollout periods selected by our strategies appropriate?** To reduce the rate of false positive, we use 0.01 as the p-value threshold for the two-sample t-tests.

Dogfood & Microsoft. In the "Dogfood" and "Microsoft (Internal)" rings, 35% of the controlled rollouts in our evaluation sample had significant movement in guardrail metrics. Of this subset of controlled rollouts, 79% of them detected significant movements within the first 3 days. The percentage change of the majority of these movements are in the range of 10%-50%. In our experience, this magnitude of difference is very challenging to be detected in such a short period by the traditional phased rollout process. Thus, controlled rollout allows feature owners to start investigating the issues and iterating **much earlier than the traditional rollout process**.

For the remaining 21% of the controlled rollouts which had significant movement in guardrail metrics, the signal was first detected on the 7-day scorecards. This verifies our claim that controlled rollouts need to run for at least 7 days (given no signal was detected earlier) in these rings. For most of the rollouts in these two rings, there were no scorecards for 14-day periods. However, we can estimate the p-values of the 14-day results by assuming the same percentage delta and

standard deviations[1] as on the 7-day scorecards, and scaling the sample size by 1.2 (based on our observation that 14-day rollouts acquire around 20% more users in these two rings than 7-day rollouts). Using this method, we predict that just 3% of the rollouts will show significant guardrail metric movements on the 14-day scorecard but not on the 7-day one. Therefore, a 14-day rollout period is not necessary for most of the rollouts. In contrast, a 7-day rollout period would be a better choice.

Insiders and Production. In the "Insiders" and "Production" rings, about 30% of the controlled rollouts had significant movement in guardrail metrics, and 61% of the controlled rollouts had significant movement in OEC or local metrics. Except for some very sensitive local metrics, the majority of these metric movements had percentage change within the ±10% interval. This magnitude of difference is difficult to be detected in the earlier rings, and almost impossible to be detected in the traditional phased rollout process by collecting user feedbacks or comparing data before and after the rollout. In contrast, controlled rollouts successfully capture these signals and provide feature owners with valuable insights.

VI. CHALLENGES OF INTRODUCING CRL

In this section, we highlight four key practical challenges that companies can expect to experience with the introduction of controlled rollout. As this was not the primary goal of this study, the list is not exhaustive. Instead, we highlight the most frequent challenges that were identified during data analysis and through authors' experience in controlled rollouts.

Controlled rollouts introduce critical dependency on data. As a result, all the challenges associated with having data in the loop will have to be solved for controlled rollouts. First, **users need to be provided value** such that they want to use their bandwidth to send data, and users need to trust that their data will be used in a manner consistent with privacy laws including the European General Data Protection Regulation (GDPR)[2]. At Microsoft, our Trusted Cloud is built on our commitments to privacy, security, transparency, and compliance. While diagnostic data may contain "personal data" as defined by Article 4 of the GDPR, all diagnostic data Microsoft collects during the use of Office applications and services, is pseudonymized as defined in ISO/IEC 19944:2017[3], section 8.3.3.

Second**, collecting comprehensive telemetry** is essential for controlled rollouts and requires building systems for collecting this data, **transforming it to a form suitable for experimentation**, and then **processing large scale data to draw meaningful insights**. These are large-scale system design challenges that need to be solved (e.g. [5]).

Third, telemetry collection is further complicated when the software is installed in a client with **variable bandwidth**, including some periods of not being connected to the internet. In these cases, the **data processing system needs to account for missing and delayed data**.

Fourth, since controlled experiments are an essential component of controlled rollout, another set of challenges involve **building the infrastructure for experimentation** [33]. For example, assigning variants – treatment and control – is a critical part of experimentation infrastructure that needs to be analytically validated before the results of experimentation can be trusted. This process, in our experience, takes multiple months. While there are tools for building basic experimentation infrastructure, a deeper integration of experimentation with the rest of the software development process is a substantial challenge.

Finally, a secondary effect of integrating experimentation with software development and deployment is the requirement of **integrating data science as a discipline in the software development team**. Introducing expertise on controlled experiments and statistics takes time and might seem to be slowing down the core tasks of the development team, but this is a necessary step to avoid pitfalls in experiment design and result interpretation [34].

VII. CONCLUSION AND FUTURE WORK

Ensuring that software meets quality standards and satisfies users' needs is critical for every company, and very challenging to get right at a large scale [34]. For this purpose, controlled experiments and phased rollouts are two novel approaches that are gaining momentum across the software product industry [3], [4], [22]. Based on our experience, however, the union of the two approaches yields the most value for product development.

In this paper, we introduced controlled rollout as a hybrid approach that combines the strength of controlled experiments with that of phased rollouts. We described the challenges of implementing and correctly applying controlled rollout, illustrating the discussion with real examples of controlled rollouts done at Microsoft. Next, we evaluated controlled rollout at MS Office and confirmed its effectiveness. Finally, we discussed practical challenges of introducing controlled rollouts into large mature software products and proposed a number of mitigations.

Since the process of controlled rollout can be viewed as controlled experimentation with different rings which may have disparate behavior, the question of what to measure at each ring and how long to run the rollout to get meaningful results is an active area of research. While our contribution proposes one answer to this question, we believe different business problems could lead to different answers. Another active area of research is on what to do when some of the rings have small sample sizes. While power analysis can provide some guidance, the problem itself is an active area of research in controlled experimentation.

Just as controlled experiments can have multiple treatments, controlled rollouts can have multiple variations and this possibility introduces new areas of research. Consider a controlled rollout with a rare bug – can this bug be detected by using multiple variations? Extending the idea

[1] This assumption may not be true for count based metrics e.g. "Number of Crashes per User", whose standard deviation increases with the length of analysis period. However, most of the guardrail metrics in our case are ratio based, e.g. "ratio of activities with crashes", whose value is between 0 and 1 and hence have much more stable standard deviation.

[2] The full document of this regulation can be found at https://eur-lex.europa.eu/eli/reg/2016/679/oj

[3] Information technology — Cloud computing — Cloud services and devices: Data flow, data categories and data use. https://www.iso.org/standard/66674.html

of multiple variations to a large number, say hundreds, of variations turns the controlled rollout process into an optimization mechanism, leading to an interesting set of questions on data pipelines and analytical processes, such as multiple comparison problems with hundreds of variants.

ACKNOWLEDGMENT

We would like to acknowledge our partners within Microsoft who have worked closely with us to evolve our thinking in this space. In particular, we would like to thank the teams in Office that works to integrate controlled rollouts into the release process. We would also like to thank everyone that shared their experiences and examples for this paper.

REFERENCES

[1] "Overview of Microsoft Windows insider program." [Online]. Available: https://insider.windows.com/en-us/how-to-overview/.

[2] J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Pearson Education, 2011.

[3] F. Auer and M. Felderer, "Current state of continuous experimentation: a systematic mapping study," in Proceedings of the 2018 44rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2018.

[4] 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 of the 2018 44rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2018.

[5] A. Fabijan, H. H. Olsson, and J. Bosch, "Customer feedback and data collection techniques in software R&D: a literature review," in Proceedings of Software Business, ICSOB 2015, 2015, vol. 210, pp. 139–153.

[6] P. Bosch-Sijtsema and J. Bosch, "User involvement throughout the innovation process in high-tech industries," J. Prod. Innov. Manag., vol. 32, no. 5, pp. 1–36, 2014.

[7] "Agile software development," Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Agile_software_development

[8] M. Shahin, M. Zahedi, M. A. Babar, and L. Zhu, "Adopting continuous delivery and deployment," Proc. 21st Int. Conf. Eval. Assess. Softw. Eng. - EASE'17, no. i, pp. 384–393, 2017.

[9] 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), 2017, pp. 18–26.

[10] J. Yli-Huumo, T. Rissanen, A. Maglyas, K. Smolander, and L.-M. Sainio, "The relationship between business model experimentation and technical debt," Softw. Business, Icsob 2015, vol. 210, pp. 17–29, 2015.

[11] 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.

[12] "Software release life cycle," Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Software_release_life_cycle.

[13] "Chrome release channels." [Online]. Available: https://www.chromium.org/getting-involved/dev-channel.

[14] E. von Hippel, "Lead users: a source of novel product concepts," Manage. Sci., vol. 32, no. 7, pp. 791–805, 1986.

[15] A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch, "Effective online experiment analysis at large scale," in Proceedings of the 2018 44rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2018.

[16] R. Kohavi and S. Thomke, "The surprising power of online experiments," Harvard Business Review, no. October, 2017.

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

[18] 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), 2017, pp. 123–132.

[19] Y. Xu, N. Chen, A. Fernandez, O. Sinno, and A. Bhasin, "From infrastructure to culture: A/B testing challenges in large scale social networks," in Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2015, no. Figure 1, pp. 2227–2236.

[20] R. Kohavi, R. Longbotham, D. Sommerfield, and R. M. Henne, "Controlled experiments on the web: survey and practical guide," Data Min. Knowl. Discov., vol. 18, no. 1, pp. 140–181, Feb. 2009.

[21] J. L. Devore and K. N. Berk, Modern Mathematical Statistics with Applications, Springer, 2011.

[22] G. Schermann, J. Cito, P. Leitner, U. Zdun, and H. C. Gall, "We're doing it live: a multi-method empirical study on continuous experimentation," Inf. Softw. Technol., Mar. 2018.

[23] 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.

[24] 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.

[25] H.-F. Hsieh and S. E. Shannon, "Three approaches to qualitative content analysis.," Qual. Health Res., vol. 15, no. 9, pp. 1277–88, Nov. 2005.

[26] M. Crotty, The foundations of social research: meaning and perspective in the research process. 1998.

[27] R. Kohavi, A. Deng, B. Frasca, T. Walker, Y. Xu, and N. Pohlmann, "Online controlled experiments at large scale," in Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '13, 2013, p. 1168.

[28] R. Kohavi and R. Longbotham, Online Controlled Experiments and A/B Testing, no. Ries 2011. Boston, MA: Springer US, 2017.

[29] H. Hohnhold, D. O'Brien, and D. Tang, "Focusing on the long-term," in Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '15, 2015, pp. 1849–1858.

[30] 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), 2016, pp. 1367–1376.

[31] G. van Belle, Statistical Rules of Thumb: Second Edition. 2008.

[32] R. Kohavi, A. Deng, R. Longbotham, and Y. Xu, "Seven rules of thumb for web site experimenters," in Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14, 2014, pp. 1857–1866.

[33] A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch, "The evolution of continuous experimentation in software product development," in Proceedings of the 39th International Conference on Software Engineering ICSE'17, 2017.

[34] P. Dmitriev, S. Gupta, K. Dong Woo, and G. Vaz, "A dirty dozen: twelve common metric interpretation pitfalls in online controlled experiments," in Proceedings of the 23rd ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '17, 2017.