2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track

Characterizing Experimentation in Continuous Deployment: a Case Study on Bing

Katja Kevic
University of Zurich
Switzerland
kevic@ifi.uzh.ch

Brendan Murphy
Microsoft Research
United Kingdom
bmurphy@microsoft.com

Laurie Williams
North Carolina State University
United States of America
lawilli3@ncsu.edu

Jennifer Beckmann
Microsoft
United States of America
jennifer.beckmann@microsoft.com

***Abstract*—The practice of continuous deployment enables product teams to release content to end users within hours or days, rather than months or years. These faster deployment cycles, along with rich product instrumentation, allows product teams to capture and analyze feature usage measurements. Product teams define a hypothesis and a set of metrics to assess how a code or feature change will impact the user. Supported by a framework, a team can deploy that change to subsets of users, enabling randomized controlled experiments. Based on the impact of the change, the product team may decide to modify the change, to deploy the change to all users, or to abandon the change. This experimentation process enables product teams to only deploy the changes that positively impact the user experience.**

***The goal of this research is to aid product teams to improve their deployment process through providing an empirical characterization of an experimentation process when applied to a large-scale and mature service.* Through an analysis of 21,220 experiments applied in Bing since 2014, we observed the complexity of the experimental process and characterized the full deployment cycle (from code change to deployment to all users). The analysis identified that the experimentation process takes an average of 42 days, including multiple iterations of one or two week experiment runs. Such iterations typically indicate that problems were found that could have hurt the users or business if the feature was just launched, hence the experiment provided real value to the organization.**

Further, we discovered that code changes for experiments are four times larger than other code changes. We identify that the code associated with 33.4% of the experiments is eventually shipped to all users. These fully-deployed code changes are significantly larger than the code changes for the other experiments, in terms of files (35.7%), changesets (80.4%) and contributors (20.0%).

***Keywords*-continuous deployment; experimentation; empirical analysis; full deployment cycle**

I. Introduction

As the software industry has moved towards a service model, different companies have adopted techniques such as continuous deployment, in which software is continually released to users [24]. Increasing the rate of releasing software, radically changes the way software is developed and deployed [5]. Previously product changes occurred as part of major releases while in continuous deployment products evolve. Some organizations have chosen to couple this rapid deployment with an experimental framework to assess the impact of changes on the end user using a practice referred to as continuous experimentation [12]. Some products, such as Bing, have been using online controlled experiments, since 2009 [15]. Visibility of continuous experimentation increased with the build-measure-learn cycles advocated in the Lean StartUp methodology [26] in 2011 based upon experiences at IMVU. As the value of continuous experimentation is more and more recognized, large organizations, such as Facebook, Google, and Netflix, increasingly employ continuous experimentation [25].

These incremental, rapid deployments offer the opportunity for development teams to formulate hypotheses about expected user behavior due to a software change, define metrics needed to be collected to verify the hypotheses, and continuously learn how users react. The process to verify hypotheses is through controlled experiments[1] [22]. Different versions of the product are exposed to randomly-chosen user subgroups. By measuring users behavior in each group, development teams have the ability to make a data-driven decision of whether to modify, delay, or abandon the given software change [15], [20], [21]. If a software change is abandoned the code associated with it, is removed from the system.

While a few works investigated the experimentation process, they often focus on the use of the process to evolve small products or services (e.g., [21]), or share experience reports and lessons learned (e.g. [17], [28]). The full life-cycle from an experiment's first code change all the way to the analysis of the captured usage measurements has not been characterized. Parts of product strategies evolve based on experiments' outcome [12], [11]. Therefore, knowing how quickly a product team can learn from experiments may help to better plan product strategies. Furthermore, knowing more about the code changes used for experiments may allow the elaboration of different experimentation procedures tailored to different kinds of code changes. Finally, we determine how many experiments are ultimately deployed to all users. Knowing more about the amount of deployed experiments helps to assess the efficiency of continuous experimentation approaches for products in different maturity stages. Previous research has not addressed how the code changes for experiments that were deployed to all users differ from the code changes which were not

[1] Also called A/B tests, split tests, bucket testing, randomized experiments, online field experiments, canary, flighting, or gradual rollouts.

978-1-5386-2717-4/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSE-SEIP.2017.19

deployed to all users. Knowing more about these differences might enable efficiencies in the product development and experimentation processes.

The goal of this research is to aid product teams to improve their deployment process through providing an empirical characterization of an experimentation process when applied to a large-scale and mature service. In particular, we investigate the following research questions:

> **RQ1**: What are the characteristics of experiments and their development efforts, in terms of time spans, number of people involved, files and changes in a large-scale and mature product?
>
> **RQ2**: What percentage of experiments are ultimately deployed to all users?
>
> **RQ3**: How do the experiments which are deployed to all users differ from the experiments which were not deployed to all users in terms of time spans, number of people involved, files and changes?

To answer these questions, we conducted a large-scale empirical analysis of Bing, Microsoft's search engine. We analyzed 21,220 experiments conducted in Bing since 2014, and all code changes that occurred during the same period of time. These experiments include a variety of different kinds of hypotheses that are tested. These hypotheses range from testing tweaks in algorithms to the impact of user interface or configuration changes on the end users. Through establishing a procedure to link specific change sets within Bing's change history to specific experiments, we analyzed the code changes committed for experiments. A change set includes one or multiple files changed at the same time. Through this analysis, we inferred whether the code changes for an experiment were ultimately deployed to all users. Change sets which we could not link to experiments were also analyzed.

The remainder of this paper is structured as follows. First, we present background on continuous experimentation and the related work which has been conducted in this area. Then, we describe how experiments are conducted within a large-scale and mature product, i.e. Bing. We describe the main points which increase the complexity of the experimentation process. Section IV describes the historical data that we used to analyze characteristics of experiments and infer whether the software change for the experiment was ultimately shipped to all users. The results of this analysis are described in Sections V,VI, and VII. We then discuss the threats to validity in Section VIII, our findings in Section IX, and conclude our work in Section X.

II. Background and Related Work

In this section, we provide background and related work on continuous deployment and continuous experimentation.

A. Background

We define and differentiate four terms used in this paper:

Continuous Integration Software is developed in smaller, incremental change sets which are regularly integrated into the codebase of the complete product, where a process automatically builds and runs a test suite daily, hourly, or even per individual change [10].

Continuous Delivery The automated implementation of an application's build, deploy, test, and release process [13].

Continuous Deployment A continuation of the continuous delivery process, where the application or service is automatically deployed to the customer [13].

Continuous Experimentation All changes require a clear hypothesis of their impact on the end customer, and that hypotheses are verified against a subset of customers prior to full deployment [12].

We found that the terms delivery and deployment are often incorrectly used interchangeably in literature on this subject.

One of the prerequisites for continuous experimentation, is that a product team deploys code changes frequently through continuous delivery or continuous deployment. Continuous integration enables both, continuous delivery and continuous deployment processes.

Through verifying the product at both, unit and system level, bugs can be detected soon after they have been introduced, and the quality of the software can be measured and analyzed over time. For example, the Apollo space mission, in the 1960s, incorporated all changes made during the day into a single overnight computer run [23]. Hence, developers can be increasingly confident about the quality of their code change. Further, by including feedback mechanisms into each step in the continuous integration pipeline, developers have the possibility to react immediately to merge conflicts, to bugs or to irregularities within the collected measures. One of the main benefits of employing the principles of continuous integration is that the product remains in a deployable state and could be released at any point in time.

Further advancements in technology beyond continuous integration enabled continuous delivery and continuous deployment practices. These later two practices originated in the Software-As-A-Service area, whereby changes to the code base could be rapidly deployed to the service and the impact of these changes on the end users can be measured.

In an experiment, different versions of the product are exposed to different randomly chosen user groups. One version of the product includes a change or a new feature, referred to as the treatment, and the other version is the current version of the product, referred to as the control [22].

For each experiment a prior hypothesis is formulated which states that the treatment is not better than the control when evaluated with a measure[2], which measures the targeted aspect of the user behavior [20]. As the experiment runs for a predefined amount of time, the initial hypothesis is evaluated through testing for statistical differences between the treatment and the control. If the null hypothesis can be rejected, the users, in fact, react differently to each version of the product.

Five main components enable developers to run experiments [20], [12], [11], [24]:

[2] Also called the overall evaluation criterion (OEC), response, dependent variable, outcome, evaluation metric, key performance indicator, endpoint or fitness function.

1) a hypothesis on the experiment's objective which is modeled in measurable metrics;
2) the instrumentation of the product;
3) a randomization algorithm;
4) an assignment method;
5) and a data path.

The product is instrumented such that the metrics defined to verify the hypothesis can be captured. The randomization algorithm is used to identify the users that are exposed to either the treatment or the control of an experiment. One difficulty for a large-scale product in which parallel experiments are run, is that the randomization algorithm has to ensure that there are no correlations between the assignments of experiments. The assignment method is the mechanism in place used to route user requests to the specified version of the product.

Users can be assigned to specific version of the product using techniques, such as traffic splitting, page rewriting, client-side assignment, and server-side assignment. Kohavi et al. [20] elaborate the advantages and disadvantages of each method. Finally, the data path is responsible for collecting the defined metrics and preparing the statistical analysis.

B. Continuous Experimentation at Microsoft

Different works analyzed the experimentation process within Bing. Kohavi and colleagues [17], [18], [19] and Crook and colleagues [6] share their insights and lessons learned while running an experimentation process at a large-scale. They work out seven rules of thumb for running controlled experiments and seven pitfalls to be avoided when running controlled experiments. They identify three main categories of challenges, including organizational challenges, engineering challenges, and the challenge of having a trustworthy experiment outcome. While Kohavi et al. [15] further look into the cultural aspects and share valuable real-world examples, Kohavi et. al [20] focus on the technical aspects in more detail and summarize the cost of experimentation when using different assignment methods. The trustworthiness of experiments is further elaborated through the analysis of five experiments' outcomes by Kohavi et al. [16]. Deng et al. [9] investigate how the percentage of users to which the experiment is exposed or the exposure duration of the experiment can be reduced while the same statistical power can be observed. Deng [8] explores an objective Bayesian A/B testing framework to analyze metrics. In this paper we build upon that work with a focus on the full life-cycle of experiments, characterizing experiment and code changes attributes.

C. Other Continuous Experimentation Research

Several case studies have been conducted which identified the challenges which are faced when employing experimentation. Other researchers [7], [15], [17], [21] have identified the cultural shifts often necessary in development teams to be one of the major challenges. In particular, the risk of individuals losing power or prestige due to experiment results contrary to their own intuitions and the importance of a consistent reward system which rewards the volume of valuable experiments regardless of outcome have been observed as the main cultural challenges. Lindgren and Münch [21] further identified that slow development cycles, the product instrumentation and the identification of the metrics to measure the user experience are further challenges. Rissanen and Münch [27] largely confirmed these challenges when they studied experimentation in a B2B environment. They further found that the capturing and transferring of user data becomes a further challenge, as legal agreements come into play.

Fagerholm et al. [11], [12] explore a model of continuous experimentation and how experiments are related to the vision and the strategy of a startup company's product. They found that the results from experiments altered the strategy of products, but the vision of the product remained unchanged. Within their suggested model, called RIGHT, the experimentation process is structured into build-measure-learn blocks. In our research, we approximate the duration of such a block.

While these case studies and experience reports focused on identifying challenges within an experimentation process and analyzed how experiments influence a product's strategy, we focus on the source code development efforts which are involved in an experimentation process.

D. Experimentation - the State of Practice.

Systematic experimentation processes are prevalent in large companies that offer SaaS services [4], [5], [17], [21], [29]. For example, at Google every change that can impact customers goes through an experimentation process [28]. Thereby, many types of changes to the product are run as experiments: from visual enhancements to changes within back-end algorithms. These companies have developed scalable platforms which offer the infrastructure to run experiments in a systematic way. Many of these advanced experimentation platforms have further tools to support the data analysis integrated. For example, LinkedIn's XLNT analysis dashboard [29] supports experimenters to make a data-driven decision of whether the experiment improved the user experience by presenting summarized views. Other tools and platforms to run systematic experiments are emerging. Google's Analytics experiment framework [1] and Facebook's PlanOut [2] are two examples of such frameworks that support an experimentation process.

When Lindgren and Münch [21] surveyed ten smaller software companies to understand the current state of the practice of experimentation processes applied, they found that the surveyed companies recognize the value of experimentation but only few companies run systematic experiments often. As more and more services and even desktop applications such as Chrome or Mozilla Firefox, adapt principles of continuous delivery [3], experimentation can become an integral part within the development cycle of a wide range of different products.

While all these case studies and experience reports enable important insights into different experimentation processes, we add to the existing body of research the first empirical study on a large-scale and mature experimentation process. In particular, compared to previous works, we describe the full life-cycle of an experiment from the first code change to the deployment of the experiment.

III. Bing Experimentation Process

For this case study, we analyze Microsoft's search engine Bing. Bing includes the main search results pages from Bing.com, as well as several services that are consumed by other Microsoft products, such as Cortana. Bing's richness in a variety of services enabled us to study the experimentation process in different environments. While the majority of the services are customer based, some are development support services for the rest of Bing (e.g. developing the deployment software). Bing is broken down into a large number of independent components, where components are either library components or dedicated to specific services. Since 2009, Bing and other services across Microsoft, increasingly use the Experimentation Platform (ExP). ExP was introduced by the Experimentation Platform team within Microsoft that was formed in 2006. ExP is a highly scalable platform that enables a systematic experimentation process [15]. In the following, we characterize the individual steps of in the experimentation process in Bing (see Figure 1).

A. Experiment Design

In a first step, developers formulate a hypothesis that defines the aspects of the users' behaviors they seek to improve. Then, they identify the set of metrics that allow the formulated hypothesis to be tested. ExP provides a wide range of predefined metrics that can be used to capture the users' behaviors. If this set of predefined metrics does not properly test the developers' hypothesis, the developers need first to implement or request the needed instrumentation within the product to capture additional aspects of the users' behaviors. The set of metrics that is identified for the experiment are then captured within an ExP scorecard. Furthermore, experimenters need to decide on the number of users that are exposed to each group within the experiment and the amount of time the experiment will be exposed to the users. A rigorous experiment design is indispensable for being able to make a data-driven decision of whether the feature should be deployed.

B. Pre-Study

Development teams have the possibility to rapidly evaluate a predetermined hypothesis by creating an internal pre-experiment prior to fully developing the software change. Internal experiments are usually mock-ups or quick-hacks of the idea that are submitted to an internal crowd-platform. Within this crowd-platform, the mock-ups or quick-hacks are shown to a chosen set of people, without identifying which is the treatment and which is the control. The outcome of these human judgments is then used to evaluate if the idea should be further implemented and then run through the full experimentation process or if the idea does not show potential. Furthermore, product teams use the outcomes of these internal experiments to prioritize the planned experiments.

C. Source Code Development and Deployment

The development team for each of the Bing services has the autonomy to choose their own software development process. Each service has its own development environment managed through its own branching structure. Also the deployment process varies among the different services and is often based on the characteristics of the service itself. For instance, the service that manages the user interface (UI) has an hourly development and deployment cycle, where the deployment process rolls out the changes in a controlled manner and rolls back changes that have bugs. Conversely, the development and deployment of complex state based services, such as the index server can require additional verification: deployment cycles can be weekly or longer.

D. Experiment Execution

After the source code is changed and deployed, the experiment execution starts. Experiments generally run for one or two weeks. To lower unforeseeable risks of system failures, an experiment generally starts by directing a small percentage of users to the advanced version, the treatment, of the product. After some time, where no failures are detected, the percentage of users directed to the treatment gradually increases. This mechanism ensures that if there was an issue with an experiment only a small percentage of users experienced it. There are different metrics which are continuously captured while the experiment is running. One group of metrics, the *guardrail* metrics, is the sentinel to the health of the product. If metrics in this group change drastically, egregious issues with the experiment are detected and ExP informs an alert system, which shuts the experiment automatically down and all traffic will be sent to the prior version. An example of a guardrail metrics is the page load time.

Since ExP allows multiple experiments to run in parallel, the risk of different experiments interacting with each other increases. Because the interaction of experiments can corrupt the metrics captured for each experiment and possibly harm the user experience with the product, it is pivotal that a possible interaction is prevented. If the prevention was bypassed, the corruption it is quickly detected. ExP incorporates mechanisms to prevent and detect interactions. To prevent interactions between different experiments, each experiment defines constraints. These constraints are used to identify the experiments that should not be exposed to the same user. To detect interactions between running experiments, ExP scans and analyzes the metrics of pairs of running experiments. If interactions between experiments are recognized, an alert is raised and the owners of the experiment involved in the interaction are informed. They then decide whether to stop one of the experiments.

If a bug in the changed code or in the experiment configuration is detected, another alert is raised which informs the owners of the experiments. If no issues are detected during the experiment execution, the experiment is stopped automatically after the exposure duration specified by the experimenter.

E. Data Analysis

To inform experimenters of the status of a running experiment, ExP allows the creation of scorecards on a periodic basis. A more extensive data analysis occurs after the experiment completed.

If the experiment ran without any issues (i.e. no alerts from the alert system reported and no bugs in the source code detected), it is considered to be a valid execution of the experiment. Developers can now decide between three alternatives: deploy the experiment, abandon the experiment, or iterate the experiment. Each experiment within Bing aims to improve user behavior on two levels. The first level is the same for each experiment within Bing. Metrics in this level are called the *main* metrics which each experiment tries to improve. These metrics target long-term goals of the product, such as the number of clicked search results. The second level is experiment-specific and targets the metrics that were defined in the product team's hypothesis for the experiment. Examples of experiment-specific metrics are the elapsed time until a first search result is clicked or whether a suggested query completion was used. Based on the product team's hypothesis and the gathered metrics a data-driven decision of whether to ship, abandon, or iterate the code change is made.

If the overall evaluation criterion (OEC) measurably improved with the new version of the product, then the treatment of the experiment is shipped and abandoned in the contrary. In practice, the OEC takes several factors into account, such as the user experience and revenue, and allows to trade one factor off for another. If the product team cannot make a data-driven decision based on the metrics that were collected, the product team iterates on the experiment design and defines a new set of metrics to test the hypothesis on. If the correct metrics have been collected for the experiment, but more user data is needed to enable a data-driven decision, then a new iteration of the experiment is launched. Finally, the product team can also decide to iterate on the source code change, but to validate the same hypothesis.

If the experiment executed with issues, then it is an invalid execution. If there was a bug in the changed code or in the experiment configuration detected, the product team iterates on a further code change to eliminate the bug. If the experiment was stopped because the metrics indicated that they were harming the user experience, then it might be abandoned.

F. Complexity of Experimentation

We observed that running a thorough experimentation process on a large scale service is very complex. Figure 1 depicts the experimentation process currently used by Bing. The complexity of the experimentation process stems from different aspects. First, while it is possible to rapidly verify that a deployment does not break the user experience, it takes time to verify that the user experience is improved or not degraded by the change. Experiments have to be exposed to the user groups for at least one week. There are many reasons for this: one reason is that users interact with the product differently on different days of the week. While it is imaginable that users search, for example, may be more work related on a Monday morning, they would rather search for social related activities over the weekend. Another reason is that it is important to have enough users for statistical validity for trustworthy comparison. Depending on the size of the change and the prominence of the feature, it takes time until a large enough number of users interacted with it to gain enough data for statistical validity. Second, since Bing has users all over the world and runs on different devices, the results of the experiment can be country and device dependent. This segmentation adds complexity to the configuration of the service. Third, there is a limited capacity to run experiments. Parallel experiments can be run for each deployment and each data center. Finally, after running a series of experiments it needs to be tested how these experiments interact with each other and whether the combined user experience is still improved.

IV. Study Data and Method

Our dataset consists of 21,220 experiments that were conducted within Bing over the last 2.5 years. Bing offers the possibility to study experimentation in inherently different components of the product. Since these different components have slightly different procedures to capture and implement experiments, the following analysis does not capture all experiments for all components within Bing.

A. Experiments

As the experimentation process within Bing emerged and changed over time, we restricted the analysis to only experiments that were created since the beginning of 2014. Our dataset comprises historical data of 21,220 experiments run within 19 components of Bing. We downloaded information about these experiments through an API offered by ExP. ExP stores attributes about experiments and stores the exposure duration of each experiment, which reflects the amount of time the experiment's treatment was exposed to end users of the product. In our analysis, we included only experiments for which a positive exposure duration was stored (occasionally experiments were created, but never run and hence the exposure duration is zero). Furthermore, ExP stores for each experiment a list of people who are responsible for the experiment, i.e. the *owners* of the experiments. Finally, we also retrieved the information showing which experiments are iterations of one another (i.e. the experiments which belong to the same experiment group).

Using another API offered by ExP, we downloaded for each experiment the created scorecards, which include several metrics measured over the experiment duration. ExP generates scorecards on a regular basis throughout an experiment. Hourly and daily scorecards measure the early hours of experiments and look for serious negative results that indicate a regression or bug in the product. As time goes on, the system

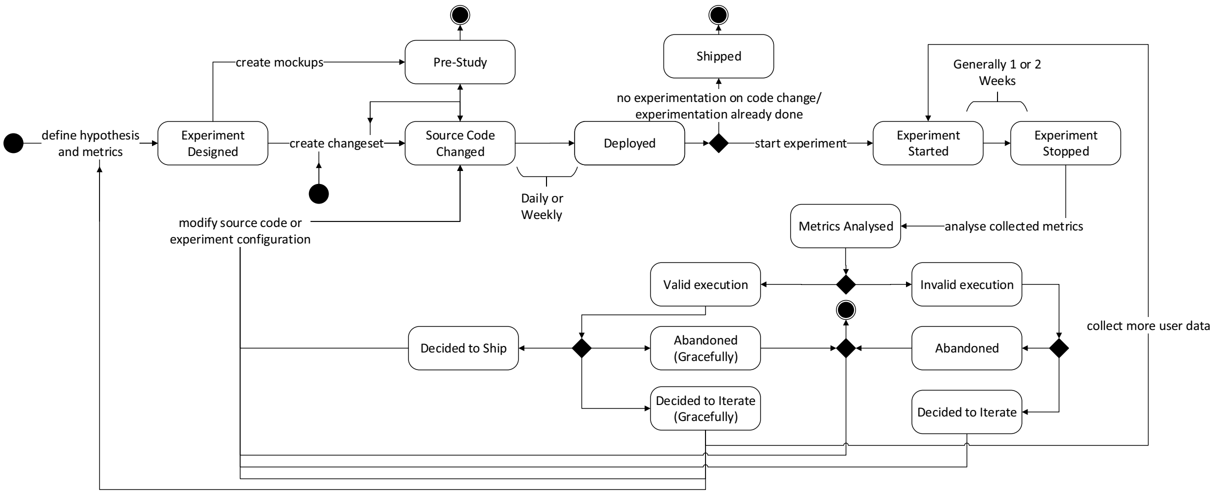

Fig. 1. Experimentation Process used in Bing.

generates fewer scorecards, because the bugs are typically detected early on, and the goal is now to determine the validity of the hypothesis. In our analysis, we considered only the last scorecard that was created for a particular experiment.

B. Linking Source Code and Experiments

No explicit link exists between experiments and the source code. To re-establish this link, we analyzed all change sets within Bing's source code change history since 2014.

Experiments in Bing are generally controlled through configuration initialization files (INI). As Bing is a large product, consisting of multiple components and developed by hundreds of developers, different syntax are used to configure experiments.

In a first step, we filtered all change sets identifying those that include the editing of at least one INI file. In a second step, we iterated through all the INI files identified in the first step and parsed the files using a regular expression to identify those INI files used to configure experiments. In a third step, we iterated through all INI files identified in the second step. We parsed each version of the files using another regular expression to identify whether the specific file version includes a configuration for one of the 21,220 experiments in our data set. Through this method, we created a link between a specific change set and a specific experiment.

As a result of this analysis, we were able to categorize and label every change occurring in the Bing development environment since 2014 into one of the following four categories:

Matched Change

The change set includes an INI file that was linked to a particular experiment.

High Probability Change

The change set includes an INI file for which we know at least one version has been used to configure experiments.

Low Probability Change

The change set includes at least one arbitrary INI file, but the INI file contains no syntax that implies it is used to configure experiments.

Other Code Change

The change set includes no INI file.

Due to the variety of complex syntax used in INI files, we concede that we may missed matched changes. However, these experiments are represented in the high probability category.

Prior to releasing the code for an experiment, a development team may iterate the code multiple times. Each code iteration is referred to as a change set. The change sets prior to the deployment of the code for an experiment are referred to as *related change sets*. To identify how much effort goes into an experiment, related change sets must be identified.

To identify the related change sets, we use the fact that a file in Bing has 1.3 iterations per year. As a result, we made the assumption that if the same file changes within a 5 day period then we can assume that the change sets containing the file are related. The small percentage (0.07%) of files that change very frequently (greater than 15 times per year) are excluded from the analysis. The following algorithm is applied to identify and process all related changes. Every change set edited by Bing since 2014 is processed, starting with the latest change set and working backwards. For each file in the change set the process identifies if the file was previously edited within the 5 day time window. If so, the change label for the change set, that the file belongs to, is altered based on the value of the label of the initial change set. If the label on the initial change set is matched change it overrides all other categories. If the label is high probability this overrides low probability and other code changes and if it was low probability this overrides other code changes. Walking backwards through the change sets will result in a cascading effect, where edits that occur within 5 days of the re-labeled change sets will also be re-labeled.

We also analyzed the identification of related changes over a time span of ten days. As the association of related changes remains roughly the same, we decided to use a time span of 5 days in this analysis.

C. Parsing the Experiment Outcome

The experiments for which we could identify one or more matched changes, allowed us to infer whether the treatment of the experiment was ultimately deployed to all users. In particular, we analyzed the sequences of changed lines (diffs) within the matched changes of an experiment.

Occasionally, the configuration names of experiments are reused. In this circumstance, we cannot infer whether the treatment of the experiment was shipped or not. The syntax for controlling the shipment of experiments' treatments is complex. The parser currently does not cover all options.

V. Experiment Characterization (RQ1)

> **RQ1**: What are the characteristics of experiments and their development efforts, in terms of time spans, number of people involved, files and changes in a large-scale and mature product?

We answer this research question from two perspectives. First, we analyze how much time an average experiment within Bing takes (Section V-A). Further, we characterize other attributes of the development efforts and of experiments, such as the people who are involved. Due to substantial differences between components of Bing, we summarize these features for each component separately in Table I. Second, we analyze Bing's change history of the past 2.5 years and compare the changes that are used for experiments to those changes not used for experiments (Section V-B).

A. Experiment Life-Cycle

Change sets for experiments are rapidly deployed. The average time between the first code change for an experiment and its deployment (last code change observed before the start of the experiment) is 1.5 days ($SD = 1.4$). Depending upon the specific component of Bing, an experiment iteration is generally exposed for one or two weeks to a user group. On average, experiments are iterated 1.8 times ($SD = 1.8$) and owned by 4.8 ($SD = 2.3$) people. On average, 1409 different metrics ($SD = 488$) are collected for an experiment. Over an experiment group, we observed 6.4 separate changes to software files submitted by 2.3 people ($SD = 1.7$). See Table I for details on the major Bing components. The analysis of the captured data and additional code changes between iterations adds additional time to the execution of the experiments. Our analysis identified that the experimentation process, from the start of the experiment to the completion of the last iteration of an experiment, takes an average of 42 days, including multiple iterations of one or two week experiment runs. Through characterizing the life-cycle of experiments, a product team is enabled to identify potential bottlenecks. Knowing where the bottlenecks are within the development cycle, enables to appoint either more resources or synchronize resources in an improved way.

B. Experimental Activity within Bing

As described in Section IV-B, we categorized each code change in Bing's change history into one of the four categories: matched change, high probability change, low probability change, and other code change. We assume that many of the changes that we could not link to experimental activity and hence were grouped into the other code changes category are tool-based changes, test related, or bug fixes. Of the changes that we could link to experimental activity, we grouped 12.1% into the matched category, 45.5% into the high probability category and 42.4% into the low probability category. We observed that changes that are related to the matched and high probability category include more files than changes which are categorized into the low probability or other change category. We observed, on average, 78.9 files ($SD = 9.6$) for the matched changes, 122.2 files ($SD = 10.3$) for the high probability changes, 37.2 files ($SD = 10.4$) for low probability changes, and 11.7 ($SD = 8.2$) files for other code changes. See Figure 2.

We also found that changes categorized as matched or high probability changes have more related changes (on average 2.0 related changes for the matched changes and 1.8 related changes for the high probability changes) than the low probability or other changes (on average, 0.6 related changes for low probability changes, and 0.6 related changes for other code changes). In summary, our analysis indicates that changes that we relate to experiments are generally larger in terms of the files changed and have more related changes.

Bing can now use these results to identify the challenges that hindered developers from launching experiments. The challenges identified can then be addressed within the experimental framework.

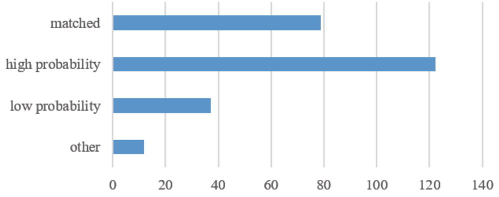

Fig. 2. Average number of files for matched, high probability, low probability, and other code changes.

VI. Success Rate of Experiments (RQ2)

> **RQ2**: What percentage of experiments are ultimately deployed to all users?

Our empirical analysis indicates that 33.4% of the experiment groups were ultimately deployed to all users. Our observation supports Kohavi et al. [15] who reported that about a third of the experiments improve the metrics they were designed to improve. For 18% of the experiment groups, our procedure cannot infer whether the experiment was deployed to all users, these would require additional analysis to identify their status (see Section IV-B for details). We also found

TABLE I

CHARACTERISTICS OF EXPERIMENTS FOR THE MAJOR BING COMPONENTS INCLUDED IN OUR ANALYSIS. THE EXPOSURE DURATION IS CALCULATED OVER ALL ITERATIONS OF AN EXPERIMENT.

Bing component	development efforts			experimentation time		
	# contributors	# files	# changes	exposure duration	# iterations	# owners
Ads	2.2	12.1	4.2	29.1	1.9	6.2
Cortana	1.8	11.7	4.2	17.3	1.6	4.7
Datamining	1.6	9.4	3.9	19.7	1.9	3.1
Engagement	1.7	13.2	4.4	30.0	2.2	4.9
Index	2.0	15.5	4.2	11.5	1.6	3.8
Infrastructure	1.2	5.6	2.2	21.8	3.1	5.0
Local	2.2	14.0	5.2	25.4	1.7	4.2
Multimedia	1.9	18.3	4.1	15.0	1.5	5.3
Relevance	2.6	13.8	7.2	16.3	1.7	4.7
Segments	2.3	9.2	5.3	23.4	1.5	3.9
UX	2.1	15.3	4.9	22.9	2.2	4.9
Windows Search	1.7	16.2	3.0	14.6	1.5	4.8

considerable differences between the components within Bing. While components related to multimedia deploy 50.7% of the experiments to all customers, the rate is lower for components related to the index server (24.9%) for example. The varying rates of deployed experiments among components in Bing indicate that different components have different levels of difficulty to innovate enhancements which significantly improves the user experience. Bing developers mentioned that they are happy that they do not have a specific target of successful experiments, enabling them to try out new ideas. We further observed that the percentage of non-deployed experiments is increasing over time. One possible root cause might be that it becomes more difficult to find a niche for innovation as the product matures. On the other hand, since ExP facilitates systematic experiments, developers might test different variants of the same feature in separately captured experiments. Through our analysis, Bing is enabled to analyze the metrics that lead to a data-driven decision. Knowing which metrics are crucial for a particular component, opens the possibility to further automate a data-driven decision and offer an improved scorecard interface.

VII. Differences between Deployed and Non-Deployed Experiments (RQ3)

> **RQ3**: How do the experiments which are deployed to all users differ from the experiments which were not deployed to all users in terms of time spans, number of people involved, files and changes?

To answer RQ3, we opposed several characteristics that we captured for the deployed and non-deployed experiments. We did not observe significant differences for the experiments' exposure durations, number of iterations conducted within an experiment group, number of experiment owners, and number of metrics collected. We found differences in the way the code is developed for an experiment. A Welch two sample t-test indicates that experiments for which the treatment was ultimately deployed to all users have significantly more changes ($M = 5.1$) associated than treatments of experiments which have not been deployed ($M = 2.9$) at the time of the analysis, $t = -15.86, p < .001$. Furthermore, significantly more people contributed these changes for the deployed experiments ($M = 2.0$) than for the non-deployed experiments ($M = 1.6$), $t = -9.05, p < .001$. We also found that the code changes for deployed experiments are overall larger, in terms of the files that were changed ($M = 22.1$ for deployed experiments, $M = 14.2$ for non-deployed experiments, $t = -11.23, p < .001$), the unique files that were changed ($M = 14.4$ for deployed experiments, $M = 10.7$ for non-deployed experiments, $t = -7.30, p < .001$), and the number of lines that were changed ($M = 690$ for deployed experiments, $M = 231$ for non-deployed experiments, $t = -2.58, p = .01$).

We can infer for experiments which were ultimately deployed to all users that the captured metrics allowed a data-driven decision. At this point of our analysis, we cannot infer for the experiments that were not deployed to all users whether the metrics indicated that the user experience is decreased or whether there was no significant difference observed between the treatment and the control. Nevertheless, our results indicate that the collaboration of more contributors leads to the fruitful execution of an experiment. To better understand whether the collaboration of more people causes more files being changed or whether the need to change more files requires more people to collaborate, is planned for future work. Understanding the differences between deployed and non-deployed experiments, teams may be able to identify which category of experiments are more likely to be more successful and which category of experiments may require more monitoring.

VIII. Threats to Validity

The *external validity* of our empirical analysis is threatened by the analysis of only one project. Because Bing is a mature large service, our results are not generalizable to less mature products. Furthermore, we also believe that the experimental process is more complex for on-premises products. However, the project which we analyzed comprises several inherently different components. We tried to mitigate this difference by considering each component separately. Furthermore, due to data consistency reasons we limited our analysis to experiments of the past 2.5 years.

The *internal validity* of our analysis is threatened by the fact that components within Bing use slightly different ways to capture, configure and deploy experiments. Bing is a composition of very large services that use different programming languages. Further, Bing is developed by hundreds of developers, who implement source code for experiments in different ways. Therefore, we were limited in the development of parsers to link code changes to experiments and to infer whether experiments have been shipped. Hence, our analysis does not cover all experiments run within Bing, but presents an analysis on a subset of Bing's experiments. Furthermore, our analysis on the experiments' life-cycle does not capture the time spent on designing the experiment and analyzing the gathered user data. Our analysis is therefore a first approximation of the actual time needed to conduct controlled experiments in a large-scale software product.

IX. Discussion

The opportunity to experiment with products drastically changed the way software is deployed within Bing. Our empirical analysis showed that experimentation has become an integral part within the deployment cycle. In the following, we discuss different aspects of the experimentation process and our planned future work.

A. Should Experimentation be Done for All Code Changes?

Bing has significantly increased the number of experiments since 2009 [17]. Further, many people are involved in the execution of an experiment who spend time preparing and executing experiments. The experimentation process is now a substantial part within the deployment cycle. We also observed that experiments can become a limiting factor of the cycle time within the deployment cycle, and hence we suggest that practice as well as research should not only focus on methods to accelerate the deployment of code changes, but on methods to identify experiments which are worthwhile to run and on methods to ensure that the experiment is run without issues.

We observed that generally larger code changes are linked to experiments. While it is possible to run a controlled experiment with each kind of change, we conclude that smaller changes have other priorities than improving the user experience. As an example, for a bug fix, the most important issues are to rapidly understand whether the deployed fix does not introduce further issues and whether the change fixes the bug. For small code changes, users may be less likely to significantly react. On the other side of the spectrum, the difficulty in the experiment design, measurement, and analysis is substantially increased for large code changes as many different aspects about the larger change can influence user behaviors. Therefore, adapting experimentation means to understand the trade-off between running a controlled experiment and other means to verify a code change or to offer different experimentation processes for different kinds of changes. We believe that the cost of a controlled experiment could be dramatically decreased for bug fixes, if these changes could be deployed after a shorter amount of time and do not have to improve the user experience necessarily. Requiring a hypothesis for every change is too great an overhead for small changes, such as big fixes. Therefore, we believe that an experimentation process tailored to the different kinds of code changes may be more efficient.

B. Size of the Code Changes.

We observed that code changes which we classified as matched changes or high probability changes have overall more development activity associated with them than changes which we classified as low probability or other changes. This observation raises the question whether experimentation in a mature system is only enabled by changes big enough to cause measurable effects on the end users of the product. On the other hand, developers may not want to spend additional time for experimentation if the change is reasonably small. Hence, we suggest that different experimentation processes and frameworks should be used for different kinds of changes. For future work, we plan to investigate further the relation between bigger releases and the amount of experiments run and compare these findings to experimentation activity in a less mature system. Furthermore, we plan to investigate how continuous experimentation influences the way developers work.

C. Developers as Data Analysts.

We observed that on average 1409 metrics about the users' behavior for each version of the product are identified and analyzed by experimenters. Further, experimentation frameworks offer to analyze an ongoing experiment multiple times a day. The collected metrics are not always straight-forward to interpret, as Kohavi et al. [16] illustrate on five real-world examples. This difficulty of interpreting the collected metrics suggests an inevitable shift of traditional development work to rigorous data analyses. This observation agrees with the observations of Kim et al. [14] who found that data scientist are increasingly important within software development teams. As these data analyses have potential to impact the annual revenue of a product [16], proper data analyses is of superior importance. As Lindgren and Münch [21] found out when interviewing people of different roles in ten software companies, a lack of time and missing expertise were named as reasons of inadequate data analysis. This lack of data analysis expertise was also identified for experiments run in a B2B environment [27]. Hence, the team around the Experimentation Platform introduced one-day classes in statistics and experiment design, which nowadays even have wait lists [15]. We plan to further investigate how developers can be supported in coping with the captured metrics about the user behavior, for example through improved user interfaces and summarized views.

D. Success of Experiments.

The success of an experiment can be considered from different standpoints. In our analysis, we first verified whether the code change is eventually shipped to all users. We observed that experiments which were shipped are significantly larger and that more people contributed source code for the experiment. However, an experiment can also be considered successful if a data-driven decision of whether to ship or abandon a code change was enabled and hence a development team had the possibility to learn more about their users. In a next step, we plan to analyze this learning value of experiments and we plan to figure out what the characteristics of experiments are that enable a data-driven decision.

X. Conclusion

The opportunity to experiment with a software product denotes a radical change in how software is deployed. While previously every change was deployed to all users, now only the changes which have a measurable improvement on the user experience are deployed to all users. We characterized such an experimentation process employed in a large-scale and mature product, i.e. Bing. We further analyzed 21,220 experiments over the past 2.5 years and observed that 33.4% of these experiments have been deployed to all users of the product. Our characterization of the experiments and their development activities revealed that experiments which are eventually shipped to all users, have generally more development activity.

Acknowledgment

We would like to thank the Experimentation Platform team for sharing historical data of experiments and helpful discussions of this work. We also thank the people who reviewed this paper, in particular the NCSU Realsearch research group, for providing valuable feedback.

References

[1] Experiments - articles & solutions. https://developers.google.com/analytics/solutions/experiments. Accessed: 2016-10-21.

[2] Planout a framework for online field experiments. https://facebook.github.io/planout/. Accessed: 2016-10-21.

[3] B. Adams and S. McIntosh. Modern release engineering in a nutshell – why researchers should care. In *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering*, pages 78–90, 2016.

[4] E. Bakshy, D. Eckles, and M. S. Bernstein. Designing and deploying online field experiments. In *23rd International Conference on World Wide Web*, pages 283–292, 2014.

[5] J. Bosch. Building products as innovation experiment systems. In *3rd International Conference on Software Business*, Lecture Notes in Business Information Processing, pages 27–39, 2012.

[6] T. Crook, B. Frasca, R. Kohavi, and R. Longbotham. Seven pitfalls to avoid when running controlled experiments on the web. In *15th ACM International Conference on Knowledge Discovery and Data Mining*, pages 1105–1114, 2009.

[7] T. H. Davenport. How to design smart business experiments. *Harvard Business Review*, pages 69–76, 2009.

[8] A. Deng. Objective bayesian two sample hypothesis testing for online controlled experiments. In *24th International Conference on World Wide Web (Companion)*, pages 923–928, 2015.

[9] A. Deng, Y. Xu, R. Kohavi, and T. Walker. Improving the sensitivity of online controlled experiments by utilizing pre-experiment data. In *6th ACM International Conference on Web Search and Data Mining*, pages 123–132, 2013.

[10] P. Duvall, S. Matyas, and A. Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. A Martin Fowler signature book. Addison-Wesley, 2007.

[11] F. Fagerholm, A. S. Guinea, and H. M˙ The {RIGHT} model for continuous experimentation. *Journal of Systems and Software*, pages –, 2016.

[12] F. Fagerholm, A. S. Guinea, H. Mäenpää, and J. Münch. Building blocks for continuous experimentation. In *1st International Workshop on Rapid Continuous Software Engineering*, pages 26–35, 2014.

[13] J. Humble and D. Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Signature Series (Fowler). Pearson Education, 2010.

[14] M. Kim, T. Zimmermann, R. DeLine, and A. Begel. The emerging role of data scientists on software development teams. In *38th International Conference on Software Engineering*, pages 96–107, 2016.

[15] R. Kohavi, T. Crook, R. Longbotham, B. Frasca, R. Henne, J. L. Ferres, and T. Melamed. Online experimentation at microsoft. In *Workshop on Data Mining Case Studies and Practice Prize*, 2009.

[16] R. Kohavi, A. Deng, B. Frasca, R. Longbotham, T. Walker, and Y. Xu. Trustworthy online controlled experiments: Five puzzling outcomes explained. In *18th ACM International Conference on Knowledge Discovery and Data Mining*, pages 786–794, 2012.

[17] R. Kohavi, A. Deng, B. Frasca, T. Walker, Y. Xu, and N. Pohlmann. Online controlled experiments at large scale. In *19th ACM International Conference on Knowledge Discovery and Data Mining*, pages 1168–1176, 2013.

[18] R. Kohavi, A. Deng, R. Longbotham, and Y. Xu. Seven rules of thumb for web site experimenters. In *20th ACM International Conference on Knowledge Discovery and Data Mining*, pages 1857–1866, 2014.

[19] R. Kohavi and R. Longbotham. Online controlled experiments and a/b tests (to appear). In *Encyclopedia of Machine Learning and Data Mining*, 2016.

[20] R. Kohavi, R. Longbotham, D. Sommerfield, and R. M. Henne. Controlled experiments on the web: survey and practical guide. *Data Mining and Knowledge Discovery*, 18(1):140–181, 2009.

[21] E. Lindgren and J. Münch. *Software Development as an Experiment System: A Qualitative Survey on the State of the Practice*. 2015.

[22] R. Mason, R. Gunst, and J. Hess. *Statistical design and analysis of experiments: with applications to engineering and science*. Wiley series in probability and mathematical statistics: Applied probability and statistics. 1989.

[23] D. Mindell. *Digital Apollo: Human and Machine in Spaceflight*. Inside Technology Series. 2008.

[24] H. H. Olsson, H. Alahyari, and J. Bosch. Climbing the "stairway to heaven" – a mulitiple-case study exploring barriers in the transition from agile development towards continuous deployment of software. In *38th Euromicro Conference on Software Engineering and Advanced Applications*, pages 392–399. IEEE Computer Society, 2012.

[25] C. Parnin, E. Helms, C. Atlee, H. Boughton, M. Ghattas, A. Glover, J. Holman, J. Micco, B. Murphy, T. Savor, M. Stumm, S. Whitaker, and L. Williams. Top 10 adages in continuous deployment. In *IEEE Software, to appear*. IEEE Computer Society, 2016.

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

[27] O. Rissanen and J. Münch. Continuous experimentation in the b2b domain: A case study. In *2nd Rapid Continuous Software Engineering*, pages 12–18. IEEE/ACM, 2015.

[28] D. Tang, A. Agarwal, D. O'Brien, and M. Meyer. Overlapping experiment infrastructure: More, better, faster experimentation. In *16th Conference on Knowledge Discovery and Data Mining*, pages 17–26, 2010.

[29] 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 *21th ACM International Conference on Knowledge Discovery and Data Mining*, pages 2227–2236. ACM, 2015.