# Continuous Data-driven Software Engineering – Towards a Research Agenda

## Report on the Joint 5th International Workshop on Rapid Continuous Software Engineering (RCoSE 2019) and 1st International Workshop on Data-Driven Decisions, Experimentation and Evolution (DDrEE 2019)

Ilias Gerostathopoulos[1], Marco Konersmann[2], Stephan Krusche[1], David I. Mattos[3], Jan Bosch[3], Tomas Bures[4], Brian Fitzgerald[5], Michael Goedicke[6], Henry Muccini[7], Helena H. Olsson[8] (organizers)
Thomas Brand[9], Robert Chatley[10], Nikolaos Diamantopoulos[11], Arik Friedman[12], Miguel Jiménez[13], Jan Ole Johanssen[1], Putra Manggala[14], Masumi Koseki[15], Jorge Melegati[16], Nuthan Munaiah[17], Gabriel Tamura[18], Vasileios Theodorou[19], Jeffrey Wong[20], Iris Figalist[21] (contributing participants)

[1]Technical University Munich, [2]University of Koblenz-Landau, [3]Chalmers University of Technology, [4]Charles University in Prague, [5]Lero, University of Limerick, [6]University of Duisburg-Essen, [7]University of L'Aquila, [8]Malmö University, [9]University of Potsdam, [10]Imperial College London, [11]Independent, [12]Atlassian, [13]University of Victoria, [14]Shopify, [15]Hitachi, [16]Free University of Bozen-Bolzano, [17]Rochester Institute of Technology [18]Universidad Icesi, [19]Intracom Telecom, [20]Netflix, [21]Siemens

gerostat@in.tum.de, konersmann@uni-koblenz.de, krusche@in.tum.de, davidis@chalmers.se, jan.bosch@chalmers.se, bures@d3s.mff.cuni.cz, Brian.Fitzgerald@ul.ie, michael.goedicke@paluno.uni-due.de, henry.muccini@univaq.it, helena.holmstrom.olsson@mau.se, thomas.brand@hpi.de, rbc@imperial.ac.uk, diamanto87@gmail.com, afriedman@atlassian.com, miguel@uvic.ca, jan.johanssen@tum.de, putra.manggala@shopify.com, masumi.koseki.wr@hitachi.com, jmelegatigoncalves@unibz.it, nm6061@rit.edu, gtamura@icesi.edu.co, theovas@intracom-telecom.com, jeffreyw@netflix.com, iris.figalist@siemens.com

## ABSTRACT

The rapid pace with which software needs to be built, together with the increasing need to evaluate changes for end users both quantitatively and qualitatively calls for novel software engineering approaches that focus on short release cycles, continuous deployment and delivery, experiment-driven feature development, feedback from users, and rapid tool-assisted feedback to developers. To realize these approaches there is a need for research and innovation with respect to automation and tooling, and furthermore for research into the organizational changes that support flexible data-driven decision-making in the development lifecycle. Most importantly, deep synergies are needed between software engineers, managers, and data scientists. This paper reports on the results of the joint 5th International Workshop on Rapid Continuous Software Engineering (RCoSE 2019) and the 1st International Workshop on Data-Driven Decisions, Experimentation and Evolution (DDrEE 2019), which focuses on the challenges and potential solutions in the area of continuous data-driven software engineering.

## Keywords

Continuous software engineering, data-driven, experimentation

## 1. INTRODUCTION

Systems we build are ultimately evaluated based on the value they deliver to their users and stakeholders. Increasing the performance of a software system or its robustness leads to faster responses to user requests and less downtime, in turn resulting in smoother user experience and higher adoption. Making a software system more intuitive and easier to use for its end-users, yet maintainable and evolvable for its developers, increases its chances of surviving in today's turbulent market. Current systems have to continue delivering value despite their fast-paced evolution due to unpredictable markets, complex and changing customer requirements, pressures of shorter time-to-market, and rapidly advancing information technologies.

To address this situation, agile practices advocate flexibility, efficiency and speed. *Rapid continuous software engineering* refers to the organizational and technical capability to develop, release and learn from software in rapid cycles, typically hours, days or very small numbers of weeks [7]. This includes determining new functionality to build, evolving and refactoring the architecture, developing the functionality, validating it, and releasing it to users [17]. The capability to perform all these activities in days or a few weeks requires significant changes in the entire software engineering approach, including parallelizing activities, empowering cross functional teams, mechanisms that allow for rapid decision making and lightweight coordination across teams. It also requires significant technical advances in the engineering infrastructure, including continuous integration and deployment, collection and analysis of post-deployment product usage data, support for running automatic live experiments to evaluate different system alternatives, e.g., A/B testing.

*Continuous experimentation* is an industry-driven approach for inferring the effect of software changes on key metrics of user acceptance [1]. It is used for improving the feedback loop between development and assessment of new features. Its main idea is often incarnated into A/B testing that compares two system variants (the one that contains a feature under test and the one that does not contain it) on their effect on important business metrics such as click-through rates and revenue [13]. This offers a systematic way for developers to hypothesize about the effect of a new feature or change and evaluate their hypotheses by collecting data from real users. Experimentation brings a number of challenges ranging from organizational and cultural ones (e.g. how to change the structure and culture of development teams to embrace data-driven practices) to technical ones (e.g. how to run different experiments in parallel and at scale) to methodological ones (e.g. which statistical method to use for comparison of different metric values). Nevertheless, it has gained great interest and enjoys large adoption by many web-facing companies including Microsoft, Google, Netflix, and Facebook.

We believe that continuous software engineering and experimentation go hand-in-hand and can jointly be used for continuous data-driven software engineering, as has been proposed e.g. in the HYPEX model [18]. The goal is to rely on data (including data collected from experiments) on different phases of the development lifecycle, which should be short in order to accommodate change. Automated testing, continuous integration, deployment, and experimentation are all emerging phases that fill-in the discontinuities that the software development life cycle presents, pushing it further toward the idea of continuous data-driven software engineering.

This new area comes with its own set of challenges, both technical and organizational, and requires deep synergies between software engineers, managers, and data scientists. This report aims to summarize the discussions that took place at the RCoSE/DDrEE 2019 workshop towards a research agenda for continuous data-driven software engineering.

## 2. RCoSE/DDrEE WORKSHOP

The RCoSE/DDrEE workshop[1] aims to bring the research communities of the aforementioned areas together to exchange challenges, ideas, and solutions to bring software engineering a step further to being a holistic continuous process. The intention is to create a highly interactive environment where different results, but also opinions and views, can be exchanged on the topics of continuous and data-driven software engineering. The overall aims of RCoSE/DDrEE are to (i) identify the problems in adoption and use of continuous software engineering and data-driven decisions, (ii) discuss new ideas that apply successful and established concepts to other domains and use cases, and (iii) build a community between software engineers and data scientists working on a common research agenda.

The first joint workshop RCoSE/DDrEE was co-located with ICSE 2019, the International Conference on Software Engineering (see https://2019.icse-conferences.org), in Montréal, Canada. The workshop attracted 9 submissions, out of which 6 were accepted. In total, there were 25 participants. The workshop started with a keynote from Jeffrey Wong from Netflix about "Mathematical Engineering in an Experimentation Platform's Measurement Ecosystem". The rest of the morning and the first session after lunch was dedicated to paper presentations.[2] The afternoon was devoted to discussion in breakout groups, where the participants focused on topics of common interest. The workshop ended with a plenary report session.

## 3. KEYNOTE

The keynote was delivered by Jeffrey Wong, Senior Modeling Architect for Netflix's Experimentation Platform (XP). At the talk, Jeffrey gave an overview of how XP is used to improve products, operations, and marketing campaigns. As an example, different artworks of a Netflix movie or series are compared to each other based on the effect they have in streaming engagement of users. The analysis of the effect of experiments is based on different causal inference algorithms, which have to be generalizable to different types of experiments and extremely scalable to be usable at the scale of Netflix (150 million users and hundreds of experiments). Jeffrey used the term *mathematical engineering* to refer to the engineering of high-performance scientific libraries for causal inference in production. Challenges in mathematical engineering include (1) programmatic ways (e.g. application programming interfaces, domain-specific languages) to describe causal effects problems, (2) generic ways to compute causal effects, and (3) scalable computation. Connecting mathematical engineering with algorithmic decision making, Jeffrey concluded with a list of remaining challenges that are high in the list of Netflix's experimentation research and include (1) maintaining controlled, randomized, environments, (2) investigating Bandit algorithms with delayed effects (e.g. how to measure the effect of an advertisement placed one month before making a movie available), and (3) making decisions when choices change.

## 4. WORKSHOP THEMES

The joint workshop focused on important and timely topics related to continuous feedback in software development and data-driven decisions. In particular, the six talks belonged to one of the following themes: providing feedback to developers, experiment-driven development and operation, and data-driven runtime decisions, overviewed below.

### 4.1 Feedback in the development life cycle

The first theme of the workshop focused on how to provide developers and managers with feedback about different aspects of the artifacts they build, including products they release. Providing quick, accurate, rich, and customer-oriented feedback is crucial in a continuous software engineering setting where each change should be evaluated based on the degree it contributes to the success of the system under development.

This theme was targeted by the first three presentations at the workshop. Focusing on security, Nuthan Munaiah proposed to use vulnerability discovery metrics, typically used to reveal engineering failures that may have led to vulnerabilities, as "agents of feedback" for the developers [16]. The presented study reports on the collection and analysis of ten metrics on six open source projects and aims to answer (i) whether vulnerability discovery metrics are similarly distributed across projects, and (ii) whether thresholds of vulnerability discovery metrics are effective at classifying risk from vulnerabilities. The work is part of the authors' vision to assist developers in engineering secure software by providing a technique that generates scientific, interpretable, and actionable feedback on security as the software evolves. Robert Chatley presented a roadmap of ideas for how new tools could be developed that harness application performance data taken from production, and present it back to developers in an actionable form [3]. He demonstrated some prototype tools that capture performance statistics from running Python web applications, calculate aggregate statistics, and use an IDE plugin to render this information in-line with the source code. With a tight development cycle, continuously deploying small changes, after a change is deployed and a number of requests have been processed in production, any effect on system performance can be visualized in the developer's IDE, giving rapid feedback. Jan Ole Johanssen et al. introduced the Continuous Thinking Aloud (CTA) approach for automating the Thinking Aloud method [11]. CTA allows developers to collect feedback during the rapid development of features. As soon as a user interacts with a new feature increment, CTA invites them to verbalize their thoughts and initiates a recording. Hereafter, CTA automatically transcribes the recordings and classifies the content into sentences of insecure, neutral, positive, or negative sentiment. CTA links the classification results to high-level changes of the feature. The results are visualized as part of a widget to support developers during the feature improvement, in particular with respect to usability.

### 4.2 Experiment-driven development and operation

The second theme of the workshop was concerned with experimentation as part of novel software development practices and operations [1]. More and more companies rely on data collected by experiments with their users in order to take business-critical decisions, e.g. whether to ship a particular version of a product [4]. Experimentation comes with its own set of challenges ranging from organizational and cultural ones (e.g. how to change the structure and culture of development teams to embrace data-driven practices) over technical ones (e.g. how to run different experiments in parallel and at scale) to methodological ones (e.g. which statistical method to use for comparing different metric values).

[1] http://www.continuous-se.org/2019

[2] The slides of all presentations (including the keynote) are available at http://www.continuous-se.org/2019/#program

Jorge Melegati [15] presented a position paper proposing hypotheses engineering, as a path to improving how hypotheses are managed in experiment-driven software engineering. In general, Bosch et al. [2] identified three approaches to software development: requirement-driven development; outcome/data/experiment-driven development; and AI-driven development. Experiment-driven software development is an approach to software development based on the use of experiments in order to build features that the user really wants, and it was influenced by the Build-Measure-Learn loop from the Lean Startup methodology.[3] Hypotheses engineering pushes for a more systematic approach to handle hypotheses in experiment-driven software development in an analogy of requirements engineering for requirement-driven software development. It consists of practices to generate, document, analyze, and prioritize hypotheses. To further develop these practices, the paper poses the following research questions:

1. How can software development teams systematically define hypotheses that need to be tested?
2. What artifact could be useful to represent hypotheses and support experiments creation?
3. How could a hypothesis artifact be used to keep experiment-useful information?
4. How could teams understand if a hypothesis can be practically tested using an experiment?
5. How could teams understand dependencies among different hypotheses?
6. How could hypotheses evolve over time?
7. Are current assumption prioritization techniques effective?
8. Could requirements prioritization techniques be adapted to hypotheses in experiment-driven development?

As an example, Kaufman et al. [12] detailed how Booking.com systemized their experimentation approach and the process they follow when conducting and documenting experiments.

Viewing continuous experimentation as a way to support runtime software evolution, Miguel Jiménez and Gabriel Tamura presented a self-adaptive and metamodel-based framework for automating the planning and execution of experiments, on a given software system. The metamodels are used to define domain-specific languages for specifying experiment designs, software structure, and virtual/physical hardware infrastructure [10]. The modeled specifications are processed and executed by a layered architecture of three feedback loops based on the DYNAMICO reference model [20]. These feedback-loops instrument the target software system under experimentation and use infrastructure-as-code to model and reconfigure the virtual hardware infrastructure in which the experiments are to be deployed and executed. In this way, the experiment design considers the application of software patterns and diverse system configurations based on the metamodels. Examples of this include the use of domain-specific design patterns (e.g., master-slave vs. producer/consumer) as variations for a part of the software architecture, variations on the input or block size to split a problem, different numbers of threads per processor core, and several configurations of distributed processes into different number of machines. The feedback loops, implementing self-adaptation properties, transform the software and infrastructure specifications into executable models (e.g., infrastructure- and configuration-as-code), and perform deployment of the software and hardware combinations for each experimental trial. While these trials are under execution, the software and the infrastructure are monitored to compute quality metrics associated with the patterns and configurations. From partial experiment results, the feedback loops can determine whether to continue or to abort the execution of each trial, as it can be resource- and time-consuming even if performed automatically, as proposed. Finally, the output from the experiment is the combination of patterns and strategies amongst the tested ones that was the best, in terms of quality metrics. In abstract, what this experimentation allows is the exploitation, at the software (re)design phase, of data obtained from the software deployed and executed in close-to-real infrastructure operation, for instance in cloud environments, thus being an example of the DevOps' shift-left realization.

[3] http://theleanstartup.com/

## 4.3 Data-driven runtime decisions

The last theme of the workshop was concerned with decisions that systems need to take at runtime based on runtime data collected and analyzed. This line of research is at the heart of research in self-adaptive systems research showcased in dedicated venues such as the SEAMS symposium and the SASO and ICAC conferences. A very interesting research challenge for our workshop is to which extent data-driven decision taken by humans can be extended to data-driven decisions taken by the systems themselves while in operation.

Focusing on this challenge and specifically on data-driven decisions for runtime deployment, Vasileios Theodorou and Nikos Diamantopoulos presented a novel approach for the elastic distribution of analytics tasks at runtime, between (1) powerful, centralized Data Centers (i.e., the cloud) and (2) Edge/Fog nodes closer to physical entities where data is actually produced (i.e., the edge) [19]. Working on the intersection of Infrastructure Virtualization, Edge Computing and Data Analysis, the authors identified and introduced the Architectural Pattern of the "Data Lagoon", as an analogous of the Data Lake at the edge, according to which data ingestion is decoupled from data processing, thus enabling rapid deployment of intelligence at Edge Gateways, as per runtime evolving requirements. This work introduced the components of a modular architecture supporting the automated spinning-up of analytics processes at the edge or their off-loading to the cloud inspired from challenging data-intensive use cases (Internet of Things (IoT) applications, Content Delivery Networks (CDN) etc.). In this direction, the authors introduced a system architecture of the Data Lagoon based on state-of-the-art technologies that can scale intelligence at run-time in dynamic and resource-limited edge environments, based on monitored QoS metrics and/or evolving application features.

# 5. OPEN RESEARCH TOPICS

The afternoon session of the workshop was dedicated to discussions in breakout groups. Each group consisted of 4-5 participants and focused on one research topic out of the four identified: "data sources, collection, and usage", "online experimentation", "education and communication", "synergies between disciplines". In the following, we provide an overview of the main findings of each group.

## 5.1 Data sources, collection, and usage

The group "data source, collection, and usage" focused on three main aspects: (1) how to combine qualitative and quantitative methods and analysis together for software improvement, (2) developers as an additional source of data, and (3) the use of continuous-* techniques not only for the improvement of user-related metrics but also for developer experience.

The discussion on the combination of quantitative and qualitative methods reinforced the need for better processes, tools and analytic systems, capable of combining instrumented data from both the system and the user behavior, with qualitative feedback given in multi-vocal sources such as blog reviews, online stores and product feedback boxes. Tools and processes that combine and streamline the analysis of both types of data can provide valuable improvement opportunities for companies and developers alike. Structured frameworks [8] and case studies [9] can inform and improve the development of such tools and processes.

The discussion on the use of developers as an additional source of data and the use of continuous-* techniques for improvement of developer experience were motivated by Robert Chatley's presentation, where field and product data are combined, passed and integrated from production to improve the development. Research has focused on

product improvement and instrumentation mainly for users. How to systematically integrate instrumentation for user improvement with instrumentation for developer needs is still an open research area.

## 5.2 Online experimentation

The discussion of the online experiment group focused on two main problems: (1) how to draw causal inference continuously and (2) how to conduct different experimental designs (as opposed to the traditional randomized controlled experiment) in software systems.

The discussion on drawing inference continuously was anchored in the keynote presentation by Jeffrey Wong. Experimentation still presents the open challenge on how to describe and compute general causal inference problems for online experiments. Specifically, the discussion reinforced the need for:

- Domain specific languages for online experiments, where engineers and scientists can formulate their problem with flexibility that goes beyond A/B testing.
- The need for better data structures and algorithms that can deal with this additional problem flexibility.
- The need to identify and separate short-term from long-term effects as well as delayed effects when a treatment is introduced in software systems.

The discussion on conducting different experimental designs was accompanied with examples from industry applications of other designs such as quasi-experiments, crossover, natural and multi-armed bandit experiments. These designs still haven't reached the level of maturity in software engineering as randomized controlled experiments, and still present open research problems on:

- How to conduct trustworthy experiments with those designs?
- What are common pitfalls and how to identify them?
- What are common situations where those designs should be explored?

## 5.3 Education and communication

The education and communication group focused on how to bring more continuous-* (continuous planning, integration, deployment, innovation experimentation, run-time monitoring) [7] aspects into the curriculum of software engineering bachelor and master programs. Specifically, it was pointed out, that there is a lack of "operations" in the curriculum in contrast to the emphasis on development aspects. Operational concepts should be introduced earlier on the curriculum to allow students at the end of their studies to feel as comfortable with them at as they are with development aspects. The group identified the following open problems:

- What is the minimal end-to-end infrastructure necessary to teach students continuous-* concepts?
- How to teach students to design and run experiments? In particular, there is a lack of tools to facilitate students to simulate users with stochastic behavior and conversion funnels so they can formulate and test their hypotheses in the duration of a course.

## 5.4 Synergies between disciplines

The group "synergies between disciplines" started a discussion about connecting business goals and metrics to product goals and metrics, and the communication needs between business-focused stakeholders (often in the role of product owners), data scientists and software engineers. The discussion suggests that data scientists or software engineers must be able to take initiative for planning and executing experiments, because they have the domain knowledge to assess costs and benefits of specific experiments. In this context, the effective and efficient collaboration between developers, operators, data scientists and software engineers was discussed. The following questions were identified to be relevant in this area:

- How can teams of developers, operators, data scientists and data engineers work together effectively and efficiently?
- How to assess the costs of experimentation, and the costs of not executing experiments?

The group elaborated about the costs involved in experimentation. Namely, time costs describe the time needed to run an experiment; adaptation costs relate to the cost of changing the system in order to run a specific experiment (e.g. cost of developing a new system variant); and convenience costs refer to costs related to customer dissatisfaction or annoyance when subjected to failing experiments. During the discussion, the group identified different ideas of what an experiment on a running system actually is, and how it deviates from experiments in a lab. Specifically, it was noted that experiments in production systems cannot have a controlled environment due to the real users, that are part of the experiment. Therefore, it is not possible to repeat the same experiment and get exactly the same results. For instance, to obtain trustworthy results, online experiments (A/B tests [13]) rely are statistical results on large number of users. The group identified the following open questions in this area:

- What exactly is an experiment in software engineering? Can this concept be generalized beyond the randomized controlled trial (A/B test) currently used in industry?
- How to manage the uncontrolled environment of user behavior in the context of an experiment?
- How could we learn from past experiments? Apart from lessons learned and common pitfalls reported by experts running experiments (e.g. [5, 14]), could we establish an experiment improvement process within a specific company?

## 5.5 Additional research topics

Beyond the topics discussed in the workshop, the evolution of rapid continuous software engineering practices introduces new organizational and technical challenges. For example, the adoption of feature flags [6] allow product teams that work on cloud products to release independently and move faster, but introduces new challenges. For business applications this poses challenges of quality control, as interaction effects between different variations across a large number of features means the number of unique product variations grows exponentially; customer support, since the support organization needs to be aware of all the changes and variations that customers can encounter and be able to respond to customer needs; and code complexity.

# 6. CONCLUSIONS

Moving towards continuous data-driven software engineering requires tackling important technical and organizational challenges. This report focused on four of them, namely on how to collect and use data for continuous software engineering, how to effectively run online experiments, how to educate students and colleagues on the concepts and techniques of continuous integration, deployment, and experimentation, and how to achieve the synergies between product owners, data scientists and software engineers.

One of the main take-aways of the workshop is that the concepts, ideas, and techniques behind continuous data-driven software engineering – short release cycles, continuous deployment and delivery, experiment-driven feature development, feedback to developers – are in great need and adoption by industry currently. This report demonstrates that they also present a wealth of important, interesting, and diverse topics for future research.

# ACKNOWLEDGMENTS

We would like to thank Jeffrey Wong for his inspiring keynote and valuable participation in the discussions. We would like to thank all participants for their contributions in the forms of papers, talks and discussions in the breakout groups. Finally, we would like to thank the

organization team of the ICSE for their efforts to build a great environment for our productive and lively workshop, the ICSE 2019 Workshops Chairs, Sven Apel and David Lo, and the RCoSE/DDrEE Program Committee comprised of Alberto Avritzer, Robert Chatley, Pavel Dmitriev, Christoph Elsner, Gregor Engels, Aleksander Fabijan, Marios Fokaefs, Wolfgang Gehring, Nikolas Herbst, Rich Hilliard, Jan Ole Johanßen, Anne Koziolek, Philipp Leitner, Wolfgang Maass, Jürgen Münch, Chris Parnin, Evangelos Pournaras, Christian Prehofer, Karen Smiley, Miroslaw Staron, Vasileios Theodorou, Bastian Tenbergen, Janek Thomas, Matthias Tichy, Andreas Vogelsang, Sebastian Voss, Xiaofeng Wang, and Danny Weyns.

# 1. REFERENCES

[1] Auer, F. and Felderer, M. 2018. Current State of Research on Continuous Experimentation: A Systematic Mapping Study. *SEAA'18* (2018), 11.

[2] Bosch, J. et al. 2018. It Takes Three to Tango: Requirement, Outcome/data, and AI Driven Development. *Proceedings of InternationalWorkshop on Software-intensive Business: Start-ups, Ecosystems and Platforms (SiBW 2018)* (2018), 16.

[3] Chatley, R. 2019. Supporting the Developer Experience with Production Metrics. *RCoSE/DDrEE 2019* (Piscataway, NJ, USA, 2019), 8–11.

[4] Fabijan, A. et al. 2017. The evolution of continuous experimentation in software product development: from data to a data-driven organization at scale. *Proc. of ICSE 2017* (IEEE, 2017), 770–780.

[5] Fabijan, A. et al. Three Key Checklists and Remedies for Trustworthy Analysis of Online Controlled Experiments at Scale. 10.

[6] Feature Toggles (aka Feature Flags): *https://martinfowler.com/articles/feature-toggles.html*. Accessed: 2019-08-30.

[7] Fitzgerald, B. and Stol, K.-J. 2017. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*. 123, (Jan. 2017), 176–189. DOI:https://doi.org/10.1016/j.jss.2015.06.063.

[8] Flaounas, I. and Friedman, A. 2019. Bridging the Gap Between Business, Design and Product Metrics. *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2019), LBW0274:1–LBW0274:6.

[9] Friedman, A. and Flaounas, I. 2018. The Right Metric for the Right Stakeholder: A Case Study of Improving Product Usability. *Proceedings of the 30th Australian Conference on Computer-Human Interaction* (New York, NY, USA, 2018), 602–606.

[10] Jiménez, M. et al. 2019. An Architectural Framework for Quality-driven Adaptive Continuous Experimentation. *RCoSE/DDrEE 2019* (Piscataway, NJ, USA, 2019), 20–23.

[11] Johanssen, J.O. et al. 2019. Continuous Thinking Aloud. *RCoSE/DDrEE 2019* (Piscataway, NJ, USA, 2019), 12–15.

[12] Kaufman, R.L. et al. 2017. Democratizing online controlled experiments at Booking.com. *arXiv:1710.08217 [cs]*. (Oct. 2017).

[13] Kohavi, R. et al. 2009. Controlled experiments on the web: survey and practical guide. *Data Mining and Knowledge Discovery*. 18, 1 (Feb. 2009), 140–181. DOI:https://doi.org/10.1007/s10618-008-0114-1.

[14] Kohavi, R. et al. 2014. Seven rules of thumb for web site experimenters. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14* (New York, New York, USA, 2014), 1857–1866.

[15] Melegati, J. et al. 2019. Hypotheses Engineering: First Essential Steps of Experiment-driven Software Development. *RCoSE/DDrEE 2019* (Piscataway, NJ, USA, 2019), 16–19.

[16] Munaiah, N. and Meneely, A. 2019. Data-driven Insights from Vulnerability Discovery Metrics. *RCoSE/DDrEE 2019* (2019), 1–7.

[17] Olsson, H.H. et al. 2012. Climbing the "Stairway to Heaven" – A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software. *2012 38th Euromicro Conference on Software Engineering and Advanced Applications* (Sep. 2012), 392–399.

[18] Olsson, H.H. and Bosch, J. 2014. The HYPEX Model: From Opinions to Data-Driven Software Development. *Continuous Software Engineering*. J. Bosch, ed. Springer International Publishing. 155–164.

[19] Theodorou, V. and Diamantopoulos, N. 2019. GLT: Edge Gateway ELT for Data-driven Intelligence Placement. *RCoSE/DDrEE 2019* (Piscataway, NJ, USA, 2019), 24–27.

[20] Villegas, N.M. et al. 2013. DYNAMICO: A reference model for governing control objectives and context relevance in self-adaptive software systems. *Software Engineering for Self-Adaptive Systems II*. Springer. 265–293.