



LUND UNIVERSITY

Understanding and Improving Continuous Experimentation

From A/B Testing to Continuous Software Optimization

Ros, Rasmus

2022

Document Version:

Publisher's PDF, also known as Version of record

[Link to publication](#)

Citation for published version (APA):

Ros, R. (2022). *Understanding and Improving Continuous Experimentation: From A/B Testing to Continuous Software Optimization*. Department of Computer Science, Lund University.

Total number of authors:

1

General rights

Unless other specific re-use rights are stated the following general rights apply:

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: <https://creativecommons.org/licenses/>

Take down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

Understanding and Improving Continuous Experimentation

From A/B Testing to Continuous Software Optimization

Rasmus Ros

Doctoral Dissertation, 2022
Department of Computer Science



LUND
UNIVERSITY

This thesis is submitted to the Research Education Board of the Faculty of Engineering at Lund University, in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Engineering.

© Rasmus Ros 2022

Department of Computer Science
Faculty of Engineering
Lund University

Dissertation 68, 2022
LU-CS-DISS: 2022-02
ISSN: 1404-1219

ISBN: 978-91-8039-177-1 (print)
ISBN: 978-91-8039-178-8 (pdf)

Printed in Sweden by Tryckeriet i E-huset, Lund, 2022

Abstract

Controlled experiments (i.e. A/B tests) are used by many companies with user-intensive products to improve their software with user data. Some companies adopt an experiment-driven approach to software development with continuous experimentation (CE). With CE, every user-affecting software change is evaluated in an experiment and specialized roles seek out opportunities to experiment with functionality.

The goal of the thesis is to describe current practice and support CE in industry. The main contributions are threefold. First, a *review* of the CE literature on: infrastructure and processes, the problem-solution pairs applied in industry practice, and the benefits and challenges of the practice. Second, a *multi-case study* with 12 companies to analyze how experimentation is used and why some companies fail to fully realize the benefits of CE. A theory for Factors Affecting Continuous Experimentation (FACE) is constructed to realize this goal. Finally, a *toolkit* called Constraint Oriented Multi-variate Bandit Optimization (COMBO) is developed for supporting automated experimentation with many variables simultaneously, live in a production environment.

The research in the thesis is conducted under the design science paradigm using empirical research methods, with simulation experiments of tool proposals and a multi-case study on company usage of CE. Other research methods include systematic literature review and theory building.

From FACE we derive three factors that explain CE utility: (1) investments in data infrastructure, (2) user problem complexity, and (3) incentive structures for experimentation. Guidelines are provided on how to strive towards state-of-the-art CE based on company factors. All three factors are relevant for companies wanting to use CE, in particular, for those companies wanting to apply algorithms such as those in COMBO to support personalization of software to users' context in a process of continuous optimization.

Popular Summary

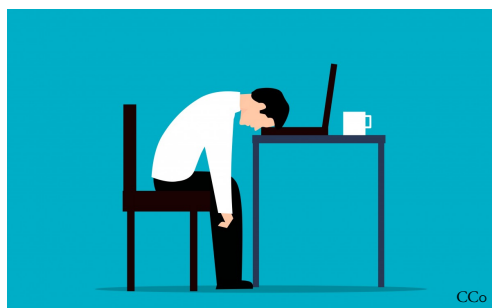
Bättre programvara med experiment

Rasmus Ros, Inst. för Datavetenskap, Lunds Universitet

ALLA har frustrerats av komplex och ointuitiv programvara som är svår att använda. Samtidigt är programvaran på t.ex. stora ehandelsbolag så enkel att använda att man med ett enda klick kan få en vara hemlevererad. Skillnaden ligger i att man inom ehandeln optimerar programvaran så att den anpassas till användares individuella behov genom kontinuerliga experiment.

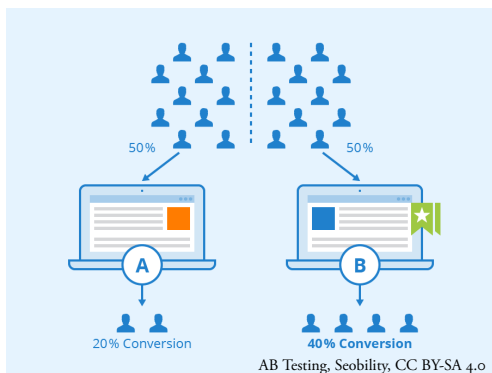
Programvara blir allt mer komplicerad att utveckla och använda. Man är dessutom allt mer beroende av programvara i sin vardag i takt med att fler industrier digitaliseras. P.g.a. den ständigt ökande mängden och komplexiteten av programvara så är det lätt att användarvänlighet får stryka på foten. Om en programvara är användarvänlig så hjälper den användaren att lösa olika problem på ett smidigt sätt. För att uppnå det så behöver programvaruutvecklarna också jobba med kontinuerlig förbättring av användarvänlighet.

En metod för att utvärdera förbättringar i programvara är experiment där man provar att lansera en förändring i programvaran för hälften av användarna samtidigt



som den andra hälften fortsätter använda en gammal variant. Genom att mäta användarmönstren för de olika programvaruvarianterna kan man utvärdera hur mycket bättre—eller sämre—ändringen blev. Den här typen av experiment med två varianter kallas vanligen A/B-test.

Experimentering möjliggörs av dagens uppkopplade värld där programvaran för olika system och enheter kan uppdateras och även övervakas via nätet så att användardata samlas in. Tillämpningen på programvara har nyligen blivit populär för företag som producerar programvara för webb-uppkopplade produkter och tjänster där mycket användardata samlas in. Dessa företag anammar kontinuerlig experimentering där alla förändringar i programvara utvärderas, även



väldigt små förändringar som användare kanske inte ens märker av. Industrijättar som Google, Microsoft, och Facebook säger att de alltid har 1000-tals experiment igång, så chansen är stor att användare varje dag deltar i flera experiment.

Vilka företag kan dra nytta av kontinuerlig experimentering?

Vi ville undersöka möjligheterna och förutsättningarna för olika företag att använda sig av experimentering. Det är inte bara ehandel som håller på med A/B-testning och kontinuerlig experimentering. Företag inom andra branscher med stora mängder användardata använder sig av det i hög utsträckning. Till exempel använder Spotify experimentering för att vidareutveckla sin musikströmningstjänst, och King använder den för att förbättra sina mobilspele som CandyCrush. Frågan är då om kontinuerlig experimentering kan användas för att förbättra användbarheten hos alla programvarusystem, även sådana med ökända användningssvårigheter, så som system för journalhantering eller tidrapportering. Vår forskning visar att svaret tyvärr är nej.

Baserat på en studie med 12 företag skapade vi en teori som förklarar orsakerna till att företag lyckas med experiment. Det kan sammanfattas med tre faktorer som berör infrastrukturen för användardata, komplexiteten av problemområdet, och typen av affärsmodell som används. Alla tre faktorer behöver beaktas för att framgångsrikt använda kontinuerlig experimentering.

Först, måste infrastrukturen för data vara på plats, det krävs oerhört mycket arbete för att samla in och analysera data från alla delar av en stor produkt. Om det saknas datainsamling på någon del av en produkt kan den delen av produktens funktionalitet inte förbättras med experimentering. Detta kan liknas vid behovet av att digitalisera i andra industrier för att kunna dra fördel av datainsamling. Således behöver även programvara digitaliseras.

För det andra, om problemet som programvaran löser för användare är komplext så är det även svårt att mäta och jämföra användarupplevelsen av olika programvaruvarianter. Till exempel, om man tittar på Kings spel så är användningen uppdelad i små spelnivåer som gör användarinteraktionen mindre komplex och således lätt att mäta.

Slutligen spelar affärsmodeller också en avgörande roll då de påverkar vilka incitament som finns för att förbättra programvara. Jämför ett programvarusystem som utvecklas för ett engångsbelopp mot en prenumerationsbaserad tjänst. För system som det betalas en engångssumma för så finns det ingen anledning att vidareutveckla det—man har redan fått betalt. Med en prenumerationsbaserad tjänst som Spotify så kommer användarna att avsluta sin prenumera-

tion om användarupplevelsen är dålig och därför finns incitament till ständig förbättring. Även att utvärdera programvaran blir lättare eftersom hur många som börjar eller sluta prenumerera på tjänsten kan mätas.

Från segmentering till personalisering

En välkänd anekdot inom A/B testningsvärlden lyder så här: om man vill ta reda på den bästa fyllningen på en pizza så kan man inte fråga en massa människor och välja en blandning av vad alla svarar, resultatet blir en oönskad sörja. Risken är alltså paradoxalt nog att man får en pizza som ingen gillar—trots att den är baserad på en undersökning av vad alla gillar. Detta är en risk med A/B testning om man tar många små beslut i rad baserat på individuella experiment.

Detta är ett argument för att anpassa programvaran till olika människors individuella behov, fast man kan behöva dela upp användarbasen i delgrupper för att praktiskt kunna hantera detta. Att utveckla separata programvaror för varje användare är inte effektivt, istället beskriver man användare med gemensamma drag som kallas segment och anpassar programvaran mot dem. T.ex. kan

en användare vara nybörjare eller erfaren användare, hen kan använda programvaran för olika syften, eller vara en betalande- eller en gratisanvändare. Tanken är då att man i varje experiment segmenterar resultaten efter dessa parametrar och ser till att varje förändring är anpassad för varje segment av användare. Med viss reservation för risker med kränkning av människors dataintegritet går det även att anpassa programvara efter ålder, könsidentitet, geografisk plats, o.s.v.

Om segmentering dras till sin spets så kan man studera kombinationer av många segmenteringsparametrar. Med tillräckligt många parametrar kan man säga att programvaran personaliseras till individnivå. Det räcker faktiskt med 46 stycken slumpmässiga parametrar för att varje person i Sverige ska kunna tilldelas ett eget unikt segment, om varje parameter enhetligt antar värdet sant eller falskt. Det krävs betydligt färre parametrar än så för att det ska bli omöjligt för en människa att överblicka eller ta beslut på resultat från experiment med sådan omfattande segmentering. Därför behövs verktyg för att stödja personalisering.

Vi har tagit fram ett sådant verktyg som kan användas för att personalisera och optimera programvara med algoritmstyrda experiment. Företag som vill applicera verktyget—eller liknande—behöver däremot välutvecklad datainfrastruktur och tillräckligt enkelt problemområde för att kunna mätas. Verktyget är utvecklat i samverkan med ett företag inom ehandel med avancerad datainfrastruktur och hög användning av experimentering. Det är släppt som öppen källkod så att andra företag kan använda det för att förbättra sin experimentverksamhet.



Acknowledgements

This work was partially supported by the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation.

I would like to extend my deep gratitude to my supervisors Prof. Dr. Per Runeson and Dr. Elizabeth Bjarnason for their constant assistance and patient advice at every stage of my Ph.D. journey. Their guidance helped me in my research and writing—but mainly to shape me as an independent researcher.

An array of thanks goes to the rest of my colleagues in the Software Engineering Research Group and the CS department at Lund University. Particularly, to Dr. Emelie Engström for the design science course that helped me frame my thesis, to Prof. Dr. Krzysztof Kuchcinski and Dr. Luigi Nardi for their patience in assisting me in technical discussions, and to Prof. Dr. Martin Höst and Dr. Alma Orucevic-Alagic for being a pleasure to co-teach under. Special thanks goes also to co-authors elsewhere; Florian Auer and Dr. Markus Borg.

Thanks also to the WASP graduate school and its leadership for arranging high quality courses and study trips; without which my thesis would be less inspired. Beyond that, thanks for laying the foundations on which I met my fellow WASP Ph.D.'s.

This thesis relies on collaboration between industry and academia. Therefore, a thanks goes out to all anonymous case companies who lent me your interviewees. I would also like to thank my former colleagues at Apptus who inspired me to pursue a Ph.D. Particularly, Dr. Mikael Hammar who helped me find my research interests.

Finally, I offer my sincere gratitude to all of my family, without their love and support none of this could have happened. Heartfelt thanks to my wife for always believing in me. Thanks also to my children for inspiring me, to my mother for encouraging me, to my sister for pushing me, and to everyone else who motivated me by engaging in my research, despite possibly understanding little.

Rasmus Ros

List of Publications

In the introduction chapter of this thesis, the included and related publications listed below are referred to by Roman numerals.

Papers Included in the Thesis

- I Controlled Experimentation in Continuous Experimentation: Knowledge and Challenges**
Florian Auer, **Rasmus Ros**, Lukas Kaltenbrunner, Per Runeson, and Michael Felderer
Information and Software Technology 134: 106551, 2021.
doi: [10.1016/j.infsof.2021.106551](https://doi.org/10.1016/j.infsof.2021.106551).

- II Continuous Experimentation Scenarios: A Case Study in e-Commerce**
Rasmus Ros and Elizabeth Bjarnason
Proceedings of the 44th Euromicro Conference on Software Engineering and Advanced Applications, SEAA, 2018. doi: [10.1109/SEAA.2018.00064](https://doi.org/10.1109/SEAA.2018.00064).

- III The FACE Theory for Factors at Play in Continuous Experimentation**
Rasmus Ros, Elizabeth Bjarnason and Per Runeson
To be submitted to a journal.

- IV Data-driven Software Design with Constraint Oriented Multi-variate Bandit Optimization (COMBO)**
Rasmus Ros and Mikael Hammar
Empirical Software Engineering 25.5: 3841-3872, 2020.
doi: [10.1007/s10664-020-09856-1](https://doi.org/10.1007/s10664-020-09856-1).

Related Publications

These papers are referenced in the introduction chapter of this thesis.

- V **Continuous Experimentation and A/B testing: A Mapping Study**
Rasmus Ros, Per Runeson
Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering, RCoSE, 2018. doi: [10.1145/3194760.3194766](https://doi.org/10.1145/3194760.3194766).

- VI **Continuous Experimentation with Product-Led Business Models: A Comparative Case Study**
Rasmus Ros, Elizabeth Bjarnason and Per Runeson
Proceedings of the 12th International Conference on Software Business, ICSOB, 2020. doi: [10.1007/978-3-030-67292-8_11](https://doi.org/10.1007/978-3-030-67292-8_11).

- VII **Automated Controlled Experimentation on Software by Evolutionary Bandit Optimization**
Rasmus Ros, Elizabeth Bjarnason and Per Runeson
Proceedings of the 9th International Symposium on Search Based Software Engineering, SSBSE, 2017. doi: [10.1007/978-3-319-66299-2_18](https://doi.org/10.1007/978-3-319-66299-2_18).

- VIII **A Machine Learning Approach for Semi-Automated Search and Selection in Literature Studies**
Rasmus Ros, Elizabeth Bjarnason and Per Runeson
Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering, EASE, 2017. doi: [10.1145/3084226.3084243](https://doi.org/10.1145/3084226.3084243).

Contribution Statement

All papers included in this thesis have been co-authored with other researchers. The authors' individual contributions to Papers I-IV are as follows.

Paper I has shared main authorship between myself and the first author, and the research was a combined continuation on papers where we each had main authorship. I conducted half of the data collection and analysis independently and wrote the background, result section, and most of the discussion in the paper (I). The supervisors had mainly a guiding and reviewing role.

I am the main author of the included papers II-IV. As such, I carried out the main part of study design, data collection, and analysis, with input and validation from co-authors. I wrote most of the sections in papers II and III and all of them in Paper IV. I was responsible for dividing the work between co-authors, the supervisors were mainly involved with giving feedback on my independent work and writing.

Paper IV was co-authored with an industry partner. All the code in the toolkit presented in Paper IV is written by me.

In addition, I have authored or co-authored four papers (V-VIII) which are related to but not included in the thesis. These are listed in the previous section.

Contents

Introduction	1
1 Background	3
2 Research Overview	8
3 Research Approach	11
4 Results	18
5 Synthesis	29
6 Discussion	34
7 Conclusions	39
Included Papers	41
I Controlled Experimentation in Continuous Experimentation: Knowledge and Challenges	43
1 Introduction	44
2 Background	45
3 Research Method	50
4 Results	54
5 Discussion	68
6 Conclusions	73
II Continuous Experimentation Scenarios: A Case Study in e-Commerce	75
1 Introduction	76
2 Method and Case Company	77
3 Results: Experimentation Scenarios	78
4 Discussion and Conclusions	82
III The FACE Theory for Factors at Play in Continuous Experimentation	85
1 Introduction	86
2 Background and Related Work	87
3 Method	94
4 Theory Formulation of FACE	104

- 5 Theory Explanations and Empirical Underpinning 112
- 6 Discussion 123
- 7 Guidelines to Practitioners for Conducting CE 126
- 8 Conclusions 129
- A Interview Guide 131
- B Code Book 134

IV Data-Driven Software Design with Constraint Oriented Multi-Variate Bandit Optimization (COMBO) 139

- 1 Introduction 140
- 2 Background and Related Work on Continuous Experimentation 141
- 3 Theory on Bandit Optimization of Software 145
- 4 Research Context and Methods 149
- 5 Tooling Support for Bandit Optimization 154
- 6 Validation 159
- 7 Discussion 168
- 8 Conclusions 173
- A Replication of simulations 174

References 175

Introduction

Understanding what software to develop is a challenging problem. The traditional requirements engineering approach [232] is to collect customer requirements before software development is started and build the software to the derived specifications. This approach is motivated in, e.g., cyber-physical systems where software must conform to hardware specifications [307]. Though, for domains where changes in users' needs and preferences are frequent, the traditional requirements engineering approach is inapt [20]. The shortcomings gave rise to user-centered agile software development [90], where the software is built iteratively to embrace change with the aid of user feedback, which is collected early and often [39]. However, much of the decision-making in agile software engineering (SE) practice is still done based only on opinions instead of facts derived from user data [171, 220].

Experimentation has been suggested as an effective and systematic method to gauge user perception of software changes [106]. A change in software can be subjected to a *controlled experiment* where the users receive either the software with the change or an old version. User data is collected for both groups and the difference between the two user groups are measured quantitatively to assess their impact. In this way, only changes that have a positive impact are accepted and delivered to all users. This type of experiment is now widespread in SE under the term A/B testing [11, V]. Experimentation is enabled by the rise of available user data and the ability to deliver changes to users through efficient continuous integration and continuous delivery tools.

Companies that use experiments to a high extent have an experiment-driven approach to software development, where software engineers or specialized roles actively search for hypotheses to evaluate in experiments. The results of an experiment might beget further questions, especially for negative results to figure out what went wrong. Thus, experiments are usually executed in a sequence, why the practice of iterative experiments is coined *continuous experimentation* (CE) [203].

CE is lauded foremost by industry-leading web-based companies such as Microsoft [172], Google [286], and Facebook [107], under the name of online controlled experiments. However, CE has been studied and been applied in industry in many different applications and domains. Examples are published in experience reports on e-commerce web shops [237, 249], mobile games [290, 321], digital libraries [322], sales tools [177], etc. Attempts have also been made to bring experimentation to less fitting domains, such as cyber-physical systems [122] and to products with business-to-business customers [242].

The practice of CE is multi-disciplinary. It spans the entire SE process and requires involvement from both software engineers—such as software developers, quality assurance engineers, and release engineers—and from specialized roles closer to business—such as business analysts, data scientists, or user experience designers. Also, experiments are driven by roles with roots primarily in different academic traditions: data scientists with quantitative methods and user experience researchers with qualitative methods. As such, studying the practice requires a multi-disciplinary approach in the boundaries between SE and other fields.

In this thesis, CE is studied and advanced from different perspectives. The main contributions of the Papers I–IV included in this thesis are:

Paper I is a systematic literature review of CE, with an analysis of the core constituents of CE, the solutions utilized to solve problems within CE, and the benefits and challenges faced when conducting CE. This paper contains an extensive overview of the body of research on CE. Industry papers on challenges and solutions dominate the field, which indicates high relevance to practitioners.

Paper II describes the scenarios that CE is used in at a case company and conceptualizes their defining characteristics. This is an early indication that experimentation is done for many reasons and that the context around experimentation matters.

Paper III presents a theory of Factors Affecting Continuous Experimentation (FACE) which can be used to explain companies' ability to conduct CE. It is based on a multi-case study with 12 case companies and 27 interviewees. Guidelines were derived from the theory concerning how practitioners can adapt to the contextual factors that affect them in order to strive towards state-of-the-art CE practice.

Paper IV introduces a toolkit called Constraint Oriented Multi-variate Bandit Optimization (COMBO) for conducting advanced experiments of software, developed in an industrial collaboration. The toolkit can be used to personalize and adapt software to a degree that is not possible using manual experiments. The toolkit uses techniques, such as, machine learning and variability management with combinatorial optimization. The paper includes an analysis on the implications to SE processes and infrastructure.

1 Background

Many web-facing companies use continuous experimentation (CE) [203] for gauging user perception of software changes [9, V]. By getting continuous feedback from users, software can evolve to meet market needs. Randomized controlled experiments in particular are emphasized by high-profile companies such as Microsoft [164], Google [286], and Facebook [107] as an evidence-based way of designing software. This section contains background information on CE in software engineering (SE) [105] and in related fields: data science and business; through the lens of experiments for product improvement, which is the topic of this thesis. CE is a multi-disciplinary practice, combining both tools and processes from continuous SE and continuous innovation practice. Finally, the section ends with tool-assisted automated experimentation for increased throughput and personalization using optimization methods.

1.1 Continuous software engineering

The drive towards continuous software updates is enabled by three practices [28, 270], see Figure 1. With (1) *continuous integration*, changes are automatically merged and integrated. This includes building an artifact, often multiple times per day. (2) *Continuous delivery* is the process by which software is ensured to be always in state to be ready to be deployed to production through testing and release packaging. Finally, with (3) *continuous deployment*, the software changes that successfully make it through the continuous integration and continuous delivery (CICD) pipeline can be deployed automatically or with minimal human intervention. Once in production, the software can be operated and monitored through automation tools that build on the CICD pipeline with *continuous runtime monitoring*. In case of failure, a rollback to an earlier version of the software can be performed. The feedback from monitoring is used to plan further coding iterations.

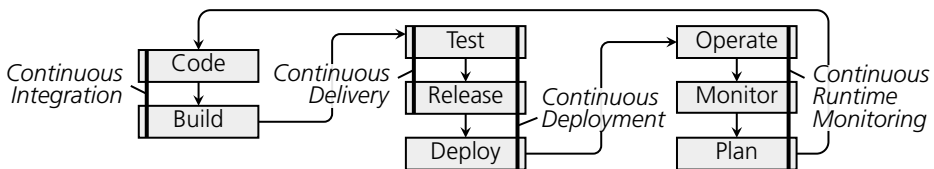


Figure 1: The eight phases of a continuous integration and continuous delivery pipeline, extended with continuous deployment and continuous runtime monitoring. The visualization is adapted from a commonly used image with unclear source [227].

Continuous deployment facilitates collection of user feedback through faster release cycles [99, 318]. With faster release cycles comes the ability to release smaller changes; the smaller the changes are, the easier it becomes to trace feedback to specific changes. This has enabled several other derived continuous SE practices—beyond continuous runtime

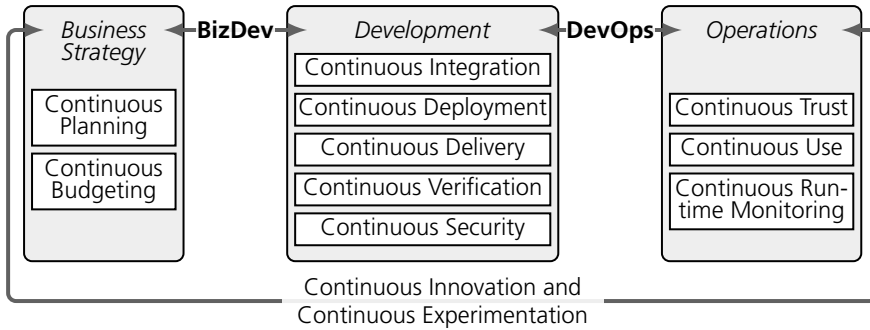


Figure 2: Adaption of the continuous * research road map for continuous software engineering by Fitzgerald et al. [113], with some practices omitted.

monitoring—as explained in the continuous * overview paper by Fitzgerald and Stol [113], for example, continuous experimentation and continuous security. A visualization of continuous * is shown in Figure 2. The value of adopting the derived continuous SE practices is perceived as high by practitioners, according to Johanssen et al. [152]. Continuous innovation and continuous experimentation, in particular, creates opportunities for collaboration by bridging all the departments involved with software engineering: business, development, and operations.

1.2 DevOps and BizDev

Successful implementation of a CI/CD pipeline should join the incentives of development and operations departments, such that developers can release often and operations get access to powerful tools to monitor and manage software. This has introduced the *DevOps* [92] concept in SE where numerous activities are automated and thereby able to be performed continuously: continuous delivery, continuous monitoring, etc. DevOps is also a role or title in some companies while at others it is used for naming the collaboration between the departments. We are also starting to see the rise of the analogous *BizDev* [113, 115, 116, 212, 303] concept that combines the responsibilities of the sales & marketing and development departments. The overlapping concept of growth hacking is a more pervasive term used by practitioners (see Section 1.3.2).

Experimentation is part of both the DevOps and BizDev concepts although for different reasons. DevOps generally do regression-oriented experiments to verify that software releases contain no changes that degrade the performance [264]. BizDev instead conduct business-oriented experiments to, e.g., increase the product’s ability to acquire customers [288]. Note that, business related metrics are usually not available in the CI/CD pipeline tools (while software performance metrics are); thus the involvement of the software development department might be necessary to conduct the wanted experiments in BizDev.

1.3 Continuous innovation

Many software companies make very incremental improvements or find niches of existing technology. *Continuous innovation* is a suitable practice for business model innovation under those circumstances [287]. Business related aspects of CE are included in Paper III. Compared to CE, continuous innovation considers all aspects of a business model, including product, processes, and organization, etc. Two concrete methodologies are popular in industry under the continuous innovation umbrella: (1) *lean startup* and (2) *growth hacking*.

1.3.1 Lean startup

Lean startup is a methodology [240] for entrepreneurship and product development that emphasises short development cycles to obtain feedback on whether a proposed business model or product is viable as early as possible. Experimentation in lean startup is predominately on prototypes rather than on completed functionality, presumably due to the immature products in startup companies. The goal of the experimentation is to find a minimum viable product (MVP), which is the smallest set of features that solve the users' problems. *Pivoting* is another method in the lean startup toolkit where the entrepreneur should realize when the current MVP is failing and break from continuous innovation and experimentation by abruptly changing direction in the business model or product.

1.3.2 Growth hacking

Products that have achieved initial success should move to a focus on growth, according to the growth hacking movement [95]. The focus is on scalable business models, where customers can be acquired rapidly once the business get going. Today, there are specialized roles for growth [96]: growth marketers, growth engineers, or growth hackers. These roles are hybrids between marketers and software engineers that work data-driven with experimentation and analysis to drive a company's user growth.

1.4 Continuous experimentation

Continuous experimentation is an overarching approach to SE [105] which considers the whole software life-cycle, from prioritization, development, to operations. The RIGHT model by Fagerholm et al. [106] describe the process and infrastructure model with inspiration from lean startup and places experimentation in a continuous SE context. The researched topics on CE in SE are covered extensively in Paper I. CE has also been studied in the human-computer interaction and data science fields under the names of: *user experience research* and *online controlled experiments*, respectively.

A *randomized controlled experiment* (or A/B test, bucket test, or split test) is a test of a *hypothesis*, where one or more variables are systematically changed to isolate the effects of the changes from influencing factors. In SE, a controlled experiment is usually used to tweak the user-facing parts of the software or to validate a new product feature with user data [V]. The metric in the experiment can be anything from the user experience (e.g. clicks), software stack (e.g. duration of a request), or sales process (e.g. conversion rate of potential customers).

1.4.1 Experiments in user experience research

Experiments have been suggested as a helpful method in the user experience (UX) research handbook [156, 257]. UX research is a methodology to systematically study how users interact with products. UX aspects are touched on in Papers III and IV but is not a major topic of this thesis. However, a majority of the published examples of experiments in CE literature make changes in the user interface [V], probably because changes in the user interface are easy to experiment on. As such, involvement by the roles UX researcher and designer is warranted in CE. Many methods for UX research are qualitative, since the focus is on solving user problems that requires knowing why users struggle with specific designs. For example, focus groups can be used on sketches or prototypes to gather detailed user opinions. These opinions can serve as hypothesis in quantitative experimentation and thus the quantitative and qualitative methods complement each other well.

1.4.2 Online controlled experiments

Controlled experiments have gained considerable interest in data science venues [I] and large web-facing companies dominate the research in the field, such as Google [286] and Facebook [128]. The seminal paper by Kohavi et al. [171] at Microsoft on CE with guidelines on controlled experiments is a recommended read. The research on online controlled experiments is focused on increased scalability, efficiency, and utility. For example, Google describes their experimentation platform for handling multiple overlapping experiments [286].

1.5 Automated experimentation and optimization

Parameterizing a range of possible values for functionality in software and having the computer pick the best value is an appealing alternative to manual experimentation. Such automated experimentation is the motivation behind Paper IV. However, it places high demands on data infrastructure and it needs a singular metric, without being able to take other metrics or qualitative data into consideration. It has still been put into practice in industry at Sentient [208] and Amazon [138]. These applications are both in the e-commerce domain. It was used for the purpose of optimizing user interfaces [208] and for personalizing the user experience [138].

The automated experimentation builds on a technique, called *multi-armed bandit optimization*. Multi-armed bandit is an experiment design suited for finding the best value of a range of values in a single parameter. It dynamically adapts the allocation of users to values, based on how well the value performs. In a classic experiment design, the allocation is statically equal for all parameter values. Multi-armed bandit is well known in industry under the term *bandit testing* [200]. The advantage is that it can obtain results faster by focusing experimentation to promising values. However, there is an increased risk of false positives, so the ability to obtain trustworthy knowledge from experimentation is lower. For multiple parameters, more advanced techniques based on combining multi-armed bandit with machine learning or a genetic algorithm can be used.

Within the last few years, advanced multi-armed bandit algorithms have been increasingly popular for optimizing behavioral algorithms online in a production environment. They can be used for tuning hyper-parameters and creating ensemble algorithms for, e.g., recommender systems [49, 301], advertisement selection systems [50], and search engines [235]. The reason for their popularity in optimizing behavioral algorithms is probably due to that these behavioral algorithms also need advanced data infrastructure; which overlaps with

the requirements from the automated experimentation. Thereby making it comparatively cheaper to implement for than for optimization of general software parameters. Optimizing software configuration settings is another use of automated experimentation in SE and has been applied to, viz., compiler flags [184] and cloud computer utilization [143].

2 Research Overview

The overall research goal of the thesis is to *describe current practice and support continuous experimentation (CE) in industry*. The research goal is further divided into three parts:

- **RG1** Map and synthesize the currently published knowledge on CE.
- **RG2** Understand current industry practice and state-of-the-art of CE.
- **RG3** Support practitioners in analysing and improving their CE practices.

The contributions of the four papers [I–IV] in this thesis are the end results of three tracks of research: literature, theory, and tool tracks. The (1) *literature track* comprised a systematic literature review (SLR) on the body of research on CE in Paper I. The investigated topics were CE processes, infrastructure, solutions, challenges, and benefits. As part of the (2) *theory track*, we identified four scenarios in which CE is used for in Paper II, and constructed a theory called FACE on factors that affect CE in Paper III. Finally, in the (3) *tool track* we designed, implemented, and validated a toolkit called COMBO in Paper IV, for supporting practitioners with automated experiments. COMBO allows experimenters to specify software parameters that are optimized with machine learning algorithms. Each of the three tracks was initiated by pilot studies, Papers V–VII, that was concluded with the Papers I, III, and IV. Paper II is also an initial study, but does not overlap with the other included papers. It provides additional context on how CE is used in practice.

Figure 3 shows an overview of the three tracks of research in the thesis and what research goals each paper addresses. The research was started by the literature track with **RG1** in mind and to provide an overview of the previous research by investigating the published literature in Papers V and I. To address **RG2**, a series of interviews was conducted at case companies involved with CE, which was published in Papers II, VII, and III. **RG3** was initiated by a tool proposition in Paper VII to investigate the feasibility of using bandit optimization algorithms to automate experimentation. This work was later expanded through additional development and validation in Paper IV through an industrial collaboration.

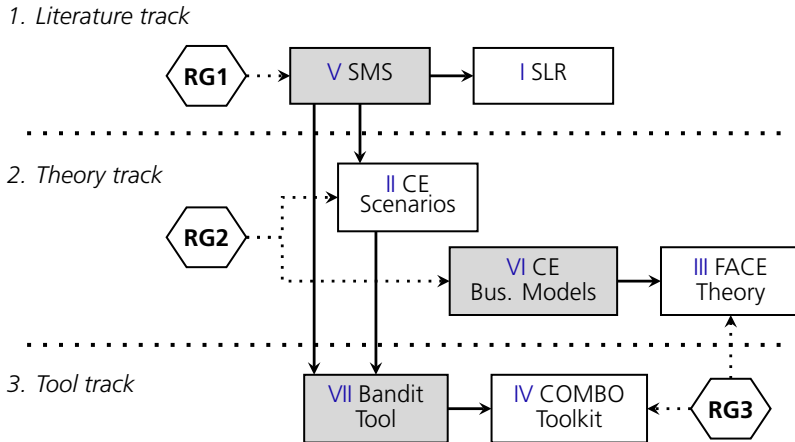


Figure 3: The three research tracks included in the thesis. Only the Papers I-IV in white boxes are included in the thesis. The *literature track* started with a mapping study and concluded with a systematic literature review. The *theory track* is based on interview material from 12 case companies and is concluded in the FACE theory. The *tool track* was initiated by a tool paper, later expanded into a toolkit through an industrial collaboration.

2.1 Literature track

The research presented in this thesis was started with the literature track of published research to address **RG1**. The first paper [V] is a systematic mapping study (SMS). The findings include which software sectors CE is used in, and the topics of the current research on CE. This paper and another SMS [9], independently published by other authors, was later deepened in a systematic literature review (SLR) [I], that expand the synthesis of the mapping studies into the core constituents of CE. Also, this SLR categorize the published solutions applied to CE, and derive the challenges and benefits of CE. Overall, the literature track provides a basis for identifying gaps in the published research, gaps that are addressed by the following two tracks.

2.2 Theory track

The theory track was started with Paper II by investigating the scenarios that CE is used in. We conducted a case study of a company that uses experiments in different parts of the company for various purposes. In particular, this study provides insights into what tool support that may be useful for what context. The automated experimentation that the tool track concerns is one such scenario. In the following paper [VI], the single case is expanded into five cases with an emphasis on comparing their business models in relation to their use of CE. These initial studies were conducted for **RG2**.

The track was concluded with Paper III, in which a theory of Factors Affecting Continuous Experimentation (FACE) was derived. FACE encompasses aspects of software engineering processes and infrastructure, complexity of the software, and business model. FACE is further expanded into guidelines for practitioners on how to adapt to the specific factors that affect their companies' CE. Thereby does Paper III contribute towards meeting both RG2 and RG3. The multi-case study that FACE is based on included more empirical data with 12 case companies in total.

2.3 Tool track

The final tool track was conducted to fulfill RG3. In both Papers V and II, it was found that automated experiments are used to tune parameters in software in a production environment. In the mapping study [V], we found that automated experiments are applied to optimize user interfaces. In Paper II, we observe that automated experiments are used to optimize the settings of a recommender system to different contexts. The general approach of automated experimentation is to define a search space that maps to the software parameters and apply an optimization algorithm, where each user is treated as a function evaluation in the optimization problem. This requires that there is a singular metric for evaluating each user session. Automated experiments were initially studied in Paper VII to investigate the feasibility of the approach. The specific approach taken in Paper VII was to combine genetic algorithms with multi-armed bandit algorithms. The same approach is reported a year later by industrial authors [208].

Paper VII was followed by Paper IV, which contains the Constraint Oriented Multi-variate Bandit Optimization (COMBO) toolkit. The toolkit was designed as part of an industrial collaboration to suit their company infrastructure, although implemented as a general toolkit. The algorithms in the toolkit are significantly improved and expanded from the one used in Paper VII, by using machine learning based multi-variate bandit optimization instead of genetic algorithms. A major contribution of the toolkit is the approach to specify configuration model parameters in an embedded domain-specific language. It supports multiple types of variables and combinatorial constraints between the variables. In this way, the toolkit expands the use cases of the technology to personalization of the software to specific user needs.

3 Research Approach

The research goals span both understanding and improving a socio-technical industry practice; which calls for a flexible approach. Therefore, this thesis employs different types of research methods. Design science (DS) has been proposed [255] as a suitable paradigm to frame prescriptive papers by illuminating different facets of contributions in empirical SE research. DS is used to frame the papers included in the thesis in Sections 4–6. The role of empirical research in DS is to evaluate solution designs (e.g. with experiments) and to study problem instances (e.g. case studies). Studies that aggregate and synthesize empirical research—secondary studies in Section 3.3—are also placed in a DS frame and serve the role to compile design knowledge or theory.

3.1 Design science

Design science is a paradigm with focus on the study of designed artifacts and the problem that they are created to solve [135, 272]. In SE research, the problem is, e.g., the design, construction, or maintenance of software and the solution artifact is usually a tool, process, or method [98]. DS is often conducted in an industry context to improve the relevance of the research. DS research is referred to as particularly suitable for prescriptive research that introduces a designed solution to a problem [255]. Though, DS can as well be used to frame studies focusing on problems only, as part of a research project that in whole studies a solution. For example, theories on design knowledge play an important role in DS to conceptualize problems such that they can be better matched with design constructs [181]. DS has also been described as a methodology [306] for conducting prescriptive research. This approach to DS is taken in Paper IV.

Cycles of problem-solving is a central concept in DS, see Figure 4. The process starts with a problem observed in practice, then an attempt at a solution is designed to address the problem based on new or existing knowledge and theories, and ending with an empirical validation of the problem-solution pair. Runeson et al. [255] describe the cycle as a series of *knowledge creating activities* between practice oriented problem-solution instances and theory-oriented problem-design constructs. The cycle consist of five such activities:

- *problem conceptualization* abstracts a problem instance to a construct;
- *solution design* maps a pair of problem-solution constructs;
- *instantiation* implements a solution construct in the real world;
- *abstraction* generalizes a solution instance to a design construct;
- *empirical validation* evaluates the aptness of problem-solution instances.

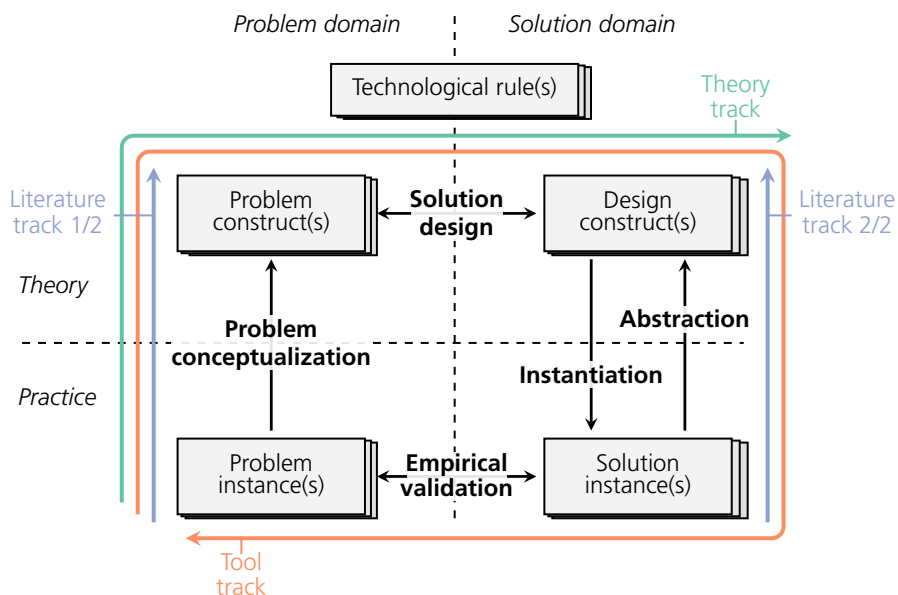


Figure 4: Design science cycle (Engström et al. [98]). The quadrants represent problem-solution pair instances/constructs and the inner arrows are knowledge creating activities (in bold). The path that each research track has taken along the cycle is shown in the outer arrows.

The contributions of an individual paper can address one or more of these activities. For example, in Paper II, we describe the observed tool support for different scenarios of CE usage. Hence the contribution is a problem conceptualization.

Using the concepts presented in Figure 4, *contributions* (i.e. design knowledge or solution artifacts) of a DS paper can be described succinctly as follows. A *technological rule* [46] is a short summary of a problem-solution pair with the template: *to achieve <Effect> in <Context> apply <Intervention>*. Hence, the technological rule is the takeaway message of prescriptive research. Storey et al. [282] further suggested a visual abstract template where the technological rule is elaborated with a description of the problem-solution pair and the approach to understanding or designing them, the relevance of the problem, the scientific rigor of the empirical validation, and the novelty of the contributions. The contributions of each included paper is discussed as such in Sections 4–6.

3.2 Empirical research

Empirical research [280] is centered around observations of phenomena which are pieces of evidence in support of an answer to a research question or goal. This thesis uses two empirical research methods: case studies and experiments. *Experiments* test a hypothesis by studying the effect of changing independent variables on measured dependent variables [310].

Experiments are used to evaluate tool proposals in Papers VII and IV with simulations. Experiments are suited to evaluate tool performance in isolation from external factors. *Case studies* are used to study a phenomenon in their real world context, with the scope limited by a specific case or cases [253]. Case studies are used in Papers II, VI, and III, to investigate CE practice at case companies. Case studies with interviews were used for their ability to answer in-depth queries contrasted with different case companies.

Further prominent empirical research methods are listed in the guidelines to select empirical research methods in SE by Easterbrook et al. [91]. In addition to case studies and experiments, they list ethnographies, action research, and surveys. The methods have different strengths and weaknesses. For example, surveys can be used to make generalizations of a larger population but lack depth and flexibility.

Observation data can be qualitative or quantitative and both types are used in this thesis for case studies and experiments, respectively. Research with *quantitative* data is usually designed to answer a more specific type of question, i.e., the relations between variables or statistical distributions of real world phenomena. Sources of quantitative data can for example be measurements of software tool performance or mining of software repositories. Statistics are used to make inference between variables in quantitative research with statistical tests in an experiment [310], regression analysis in data mining, etc. Research with *qualitative* data is flexible in the type of question that can be answered and enables gaining a deeper and broader understanding of phenomena in a context, e.g., identifying business-related factors that affect CE practices. Qualitative data is text or transcribed audio or video recordings. This data can be organized by coding pieces of text relevant to a research question with a shorthand that can then be the basis of further analysis. The primary approach to qualitative analysis taken in this thesis is to organize the codes into themes, which is referred to as thematic analysis [68]. Codes are pre-defined in a code book in this thesis, but can also be created on the fly in exploratory coding.

3.2.1 Experiments

An experiment is used to investigate the causal relation between variables with quantitative data. This requires careful isolation from confounding effects in a regulated environment. For example, to study whether a specific tool can improve the performance of software developers with respect to a given metric, then the researchers can design an experiment in which participants can solve a task with or without the tool. Controlled experiments are often considered the golden standard of science [169], but they can only be used to answer questions that can be controlled. Note that the experimentation process in science is very similar to the engineering experimentation used in CE, as presented in Section 1.4.

The use of controlled experiments for research has a long history [112], and there are now many variants of experiments [136], e.g., experiments with multiple variables (multi-variate tests) or experiments where the experiment control is simplified by being done sequentially (natural experiments).

Wohlin et al. [310] provide guidelines on how to conduct experiments in SE. They draw a distinction between *technology oriented* and *human oriented* experiments, where the difference is the type of subjects. This thesis uses technology oriented experiments to evaluate the computational performance and aptness of tools. The technology oriented experimentation in Papers VII and IV are referred to as *simulation experiments* since they simulate the usage of real users in a lab environment instead of a production environment.

3.2.2 Case studies

Case study methodology is suitable for studying real world phenomena in their original context. The case studies in this thesis follow the guidelines by Runeson et al. [253] and are based on practitioner interviews with thematic analysis [68]. Case studies and experiments are on the opposite ends of the control versus realism spectra [279]. In a case study, the context is embraced and studied as part of the phenomena rather than attempting to isolate the context from the phenomena. A *case* is defined as the bounding box of the study, it could be an organization, a team within an organization or a tool, a role etc. The *unit of analysis* in a case study is the subject of the research. Using multiple sources of data, referred to as *triangulation*, is encouraged to increase the ability to make generalizations and to separate the context of the case from the unit of analysis. Triangulation can be achieved by using multiple research methods or by studying multiple cases simultaneously in a multi-case study, which is the approach taken in this thesis. Finally, in a comparative case study the emphasis is on systematic differences between multiple cases using cross-case analysis [69], which is done in Paper VI.

3.3 Secondary studies

Research can be performed through primary as well as secondary studies. While a *primary study* collects empirical data directly from the real world with empirical research, a *secondary study* aggregates or synthesizes research from primary studies.

3.3.1 Systematic literature studies

A systematic literature review (SLR) is a secondary study with the goal of aggregating several empirical research studies. The guidelines by Kitchenham et al. [169] are used in this thesis. SLRs are conducted to answer a specific set of research questions. As such, they require a sufficiently mature research field, so that there are multiple studies on the same narrow topic. SLRs have an increasingly important role in SE research as the body of empirical research grows [169]. An SLR is conducted with a rigid search and selection process to find primary studies in a way that reduces bias and enables replication. The SLR process also includes a quality evaluation of the search, selection, and analysis. Automation of SLR search and selection is an active area of research in SE [291, VIII], but the practice still involve a lot of manual work.

A systematic mapping study (SMS) is a simplified variant of systematic literature studies with open-ended research questions. It is conducted to provide a map (or taxonomy) of the conducted research, in terms of topics, companies, research groups, etc. A SMS is often done as a pilot study of a more comprehensive SLR, as is the case in this thesis with Papers V into II.

3.3.2 Theory building

Finally, theory building is a method to generalize and synthesize research into a more mature, condensed form. Theory building is viewed as a secondary study here, irregardless of whether the data that the theory is based on is published separately or not. In theory-oriented SE by Stol and Fitzgerald [278]—see Figure 5—they describe how theory and empirical research interplay; *observations* are first abstracted into *empirical generalizations* and then further processed with *formal theory formation*. According to Stol and Fitzgerald [278], the value of a theory is twofold: (1) to anchor the research by providing insights that *informs the study design* of new studies, thereby forming a cycle of research and (2) to use as a basis to *derive* practitioner guidelines from. Making guidelines from empirical generalizations is seen as a *shortcut*, with the risk of them being too specific to observations. In the theory track, Papers II and VI are on the level of empirical generalization. In Paper III, the conceptualization step to a formal theory formation is taken and used to derive guidelines from.

There are many different definitions and meanings of theory in SE research and in other fields, e.g., in social sciences [1]. Wieringa et al. [304] identify six strategies for building theories in SE, the strategies also result in different types of theories. For example, theories built from quantitative experimentation data might be used for causal predictions but might not be feasible to create for socio-technical phenomenon. Case-based strategies is

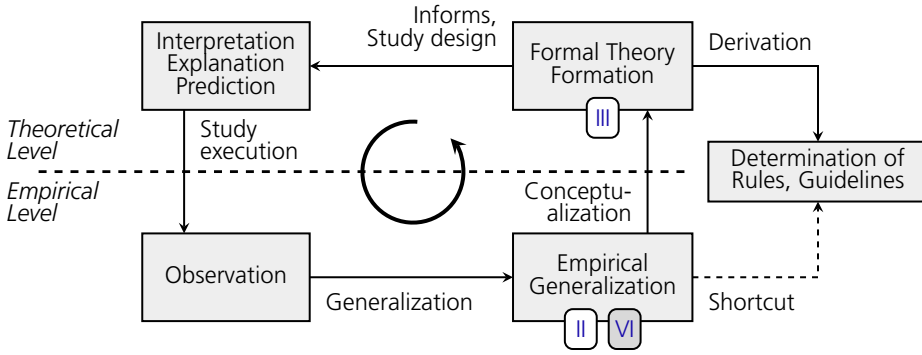


Figure 5: Theory-oriented SE (Stol and Fitzgerald [278]), on the role of theories in SE. The Papers in the theory track, II, VI, and III, are placed in small boxes to indicate their conceptualization level.

the approach used in this thesis, it is based around conceptualizing from multiple cases. Grounded theory [124] is another prominent approach in SE [4, 281] which comprises a full research process, where the researchers starts without preconceived notions about previous theory or related work (at least in the classical variant).

We use the process guidelines by Sjøberg et al. [274] on how to generalize and structure theory from cases with empirical data. Their process uses elements of grounded theory. A theory as defined by them consists of concrete constructs that the theory makes statements about in the form of propositions, which are relations between the constructs. In Paper III, the analysis is a qualitative coding step from case study interview material. Codes and themes were constructed and updated in parallel and as new theory concepts were discovered they were compared to the existing ones in a process of constant comparisons [68]. A theory is also only valid in a certain scope, based on the scope of the underlying empirical data.

3.4 Classification of papers

Each paper in the research project is classified according to what main *empirical research method* they used and what DS oriented *contribution type* they contain. The contribution types use the category clusters introduced by Engström et al. [98] where each category corresponds to a common path along the DS cycle (c.f. Figure 4), with one or more of the *knowledge creating activities* [255]. The contribution types [98] are:

- *problem-solution pairs* complete a DS cycle from problem to validated solution;
- *solution validations* focus on *empiric validation* of a solution instance;
- *solution designs* are *abstractions* to a construct with an implicit problem;

Table 1: Classification of the papers in the research project based on Engström et al. [98]. Rows in gray indicate related pilot studies not included in the thesis.

Paper	Contribution type	Methodology
<i>Literature track</i>		
Paper V	Meta	Systematic mapping study
Paper I	Meta	Systematic literature review
<i>Theory track</i>		
Paper II	Descriptive	Case study
Paper VI	Descriptive	Comparative case study
Paper III	Design theory	Multi-case study, Theory building
<i>Tool track</i>		
Paper VII	Solution design	Simulation experiments
Paper IV	Problem-solution pair	Design science methodology, Simulation experiments

- *descriptive* papers describe problem constructs with *problem conceptualization*;
- *meta* papers are aimed at researchers and can take any path along the DS cycle.

Paper III does not fit neatly into any of the contribution types of Engström et al. [98]. This is not surprising as their contribution types are based on observations from a single research venue (ICSE best papers). The theory in Paper III is a problem construct in the context of a well known design construct (i.e. CE), it also contains guidelines which is another design construct. However, the paper does not go into a full cycle by validating the solution in industry. As such the following contribution type is added:

- *design theories* have extensive conceptualization of problem instances in relation to a design construct and may also include recommendations or guidelines.

Table 1 presents an overview of the classifications. The papers in the literature track [V, I] are both meta studies from a DS perspective, but they both include the parallel activities of problem conceptualization and solution abstractions (shown as separate arrows in Figure 4). The three studies in the theory track [II, VI, III] are focused on describing problem instances into more generalized problem constructs and are *descriptive* and *design theory* papers. In the tool track, the pilot study [VII] contains a solution with limited problem context, hence the contribution is a *solution design*. In the final paper [IV], the solution is anchored with an industrial collaboration and considers the whole DS cycle with a *problem-solution pair*.

Table 2: DS abstract for Paper I:
Controlled Experimentation in Continuous Experimentation: Knowledge and Challenges.

TRs	<i>Technological rules for each solution theme are included in Paper I, Section 5.2.</i>
Problems	CE was reported to be in use at many large web-facing companies and the way these companies have structured their processes and infrastructure is well documented (RQ1). However, the challenges with CE is multi-faceted and range from business and organizational challenges to statistical and technical challenges (RQ3).
Solutions	The solution papers were focused on how to obtain more value from CE (RQ2), for example, by obtaining additional insights, richer experiment feedback, higher throughput of experiments, etc. Few papers were on overall benefits of CE (RQ4); which is principally to increase product quality. Other benefits include, e.g., improved prioritization and reduced product scope.
Relevance	There is a high degree of industry authors in CE, publishing primarily experience reports and solution designs, which indicates a strong industrial interest in the area.
Rigor	The research procedure follow established guidelines [169]. All steps of the literature search and selection process were cross-validated by different co-authors.
Novelty	The paper presents a systematic and comprehensive synthesis on CE use in industry.

4 Results

This section summarizes the papers included in the thesis. For each paper, the main contributions are summarized as *DS abstracts* based on the concept of visual abstracts in design science as described by Storey et al. [282]. The three parts of the DS abstracts are: (1) the technological rule (TR) which is a summary of the paper; (2) the problem-solution pair the paper presents, and (3) an overview on the research quality in terms of industrial relevance, research rigour, and research novelty.

4.1 Paper I: CE SLR

The goal of Paper I was to synthesize the available research on CE. Four research questions were posed: **RQ1** *What are the core constituents of CE?* **RQ2** *What solutions are available in CE?* **RQ3/4** *What are the benefits and challenges of CE?* To that end we conducted a comprehensive systematic literature review (SLR) following an established procedure by Kitchenham et al. [169]. An SLR is a secondary study since it combines multiple primary studies. The DS abstract for the paper is provided in Table 2. The prescriptive parts (TRs and problem-solution pairs) are summaries of the primary studies in relation to the research questions.

The review is an extension of two independently conducted mapping studies. Both prior mapping studies were done roughly at the same time and with a similar selection strategy and inclusion criteria but by different researchers. We combined these prior studies into a joint review by extending the sets of primary papers in both studies with an independently conducted forward snowballing that ended up with 128 selected papers. The primary papers were analyzed with thematic and narrative analysis. Figure 6 shows an overview of the identified themes and sub-themes and is organized into the four research questions.

The main findings of the literature review per research question follows. **RQ1** The required CE framework includes experimentation process and infrastructure. There are numerous experiment reports with real lessons learned caused by CE failures. For example, one reported mistake is to conduct experiments with too large scopes which makes negative results costly. For this reason, and others detailed in Paper I, it is important to follow a rigorous experimentation process. The infrastructure needs to get started with CE are reported as modest; the biggest hurdle seems to be organizational and business aspects. **RQ2** The solutions proposed in the primary papers are grouped and mapped to a technological rule with a problem-solution pair to make it clear what the purpose of each solution is. The solutions are also related to what stage of the CE process they are applied in, from ideation to analysis. **RQ3** The identified challenges were both many and spanning the entire software engineering process and organization. **RQ4** The benefits were mostly implicit or mentioned in passing as improving software quality in a metric of choice. Also, the primary papers include a large number of industry authors which indicates high relevance and that the challenges might be worth to overcome.

Finally, the paper contain research directions for future work and gaps in the research. Other researchers have filled one of the identified gaps in research by providing data sets for CE [189]. Our efforts are in unifying proposed design constructs and models, in Paper III by basing the theory building on related work and in Paper IV by combining techniques for automated experimentation with variability management techniques.

4.2 Paper II: The CE Scenarios

Paper II is a case study with interviews at the Swedish business-to-business (B2B) company Apptus. They have an e-commerce platform with data-driven algorithms that are in use at client customers' web shops. There, experimentation is used for different purposes, with different tools, and in different parts of their software product. The research was thus conducted to answer the following question: **RQ** *In what scenarios are experiments used and what tools exists to support those?* Four scenarios were identified, named after the characteristics that differentiate them and are illustrated in Figure 7. The scenarios for experimentation

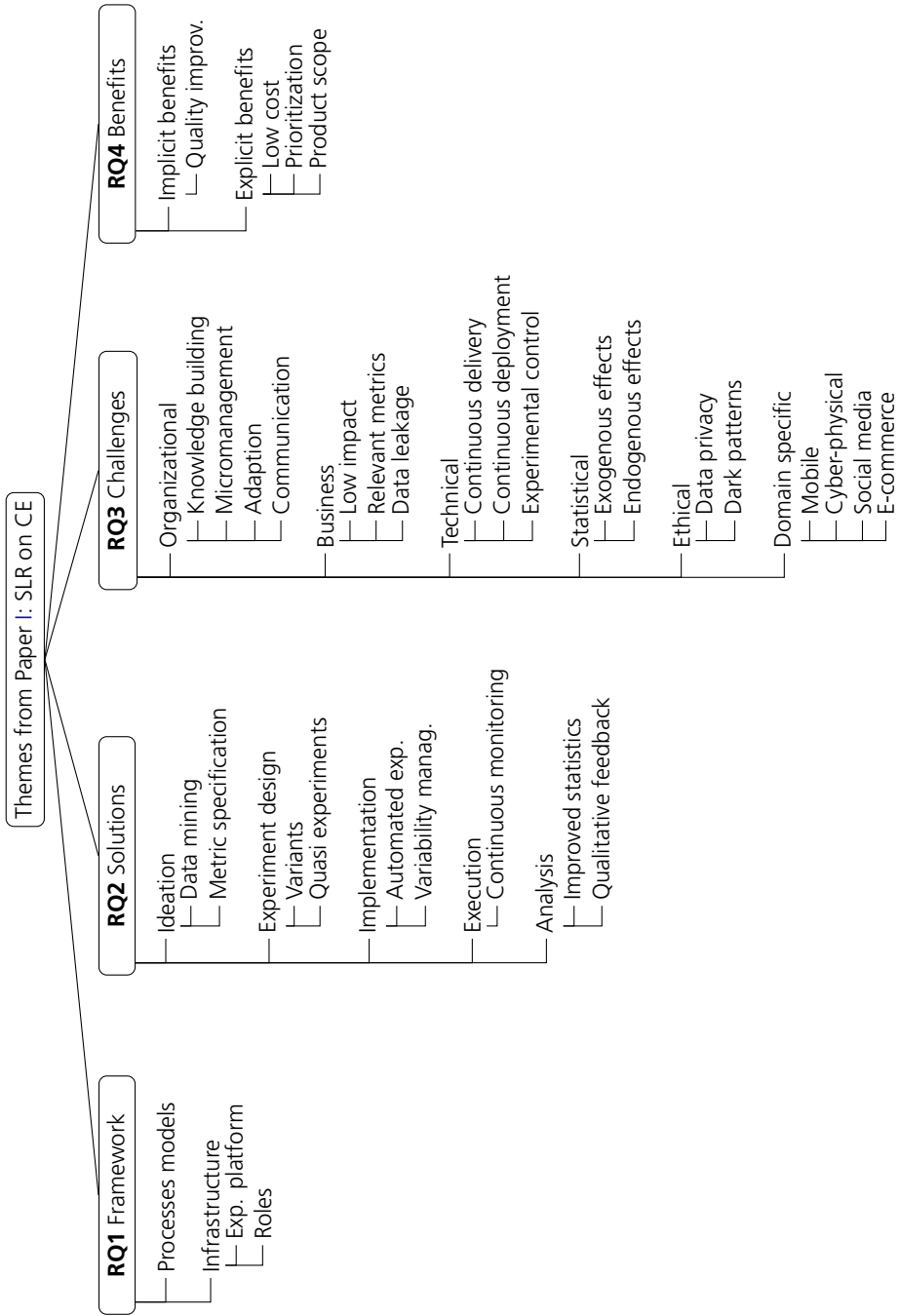


Figure 6: The resulting themes of the thematic analysis in Paper I organized per research question.

Table 3: DS abstract for Paper II:
Continuous Experimentation Scenarios: A Case Study in e-Commerce.

TR	To achieve <i>sufficient tool support for CE</i> in companies with diverse experimentation consider <i>how experimentation is used and for what purpose</i> .
Problem	Experimentation was found to be used in four different scenarios at a case company and four characteristics are derived that differentiate them: <i>purpose</i> (optimization or validation), <i>automation</i> (automated or manual), <i>control</i> (internal or external), and <i>recurrence</i> (singleton or repeated). The tool support in the scenarios varies from none to extensive.
Solution	N/A
Relevance	The case company employees are experts on CE; through experimenting on their product and by consulting on their e-commerce clients' experimentation. Experimentation is widespread in e-commerce [V].
Rigor	The scenarios were derived from material with five interviewees following guidelines for case study research.
Novelty	The scenarios are the initial step to a full taxonomy of CE.

are especially clear due to the case company's B2B relation to their customers, since experimentation crosses organizational barriers at the case company. The results in the paper are not limited to B2B companies, the scenarios have all been observed at other companies in Paper III. See Table 3 for the DS abstract of the paper; the paper does not feature a solution.

The main contributions of the paper are twofold. First, the paper includes a description of four observed scenarios of experimentation and a problem construct with the differentiating attributes between the observed scenarios. Second, an analysis of the observed tool support for the four scenarios. While the degree of tool support might not generalize to other companies, it is an early indication that the more complexities there are in the context surrounding an experiment, the harder it is to conduct experiments and the less insights can be obtained from the experiment results. This line of inquiry form the basis for Paper III.

The scenarios also show that experimentation is conducted in all three of the sales & marketing, software development, and operations departments. Though, these departments have differing goals in their experimentation, the sales & marketing departments are involved with optimization of the sales funnel, the software development and operations departments do validations of new features or changes. The tool support for the scenarios also differ, the support for optimization at the sales & marketing department is particularly extensive and easy to use. While the tools for the software development department's validation experiments are flexible to be able to support different types of development and in different parts of the software. Finally, the tools for operations was lacking due to how rare the Scenario c was.

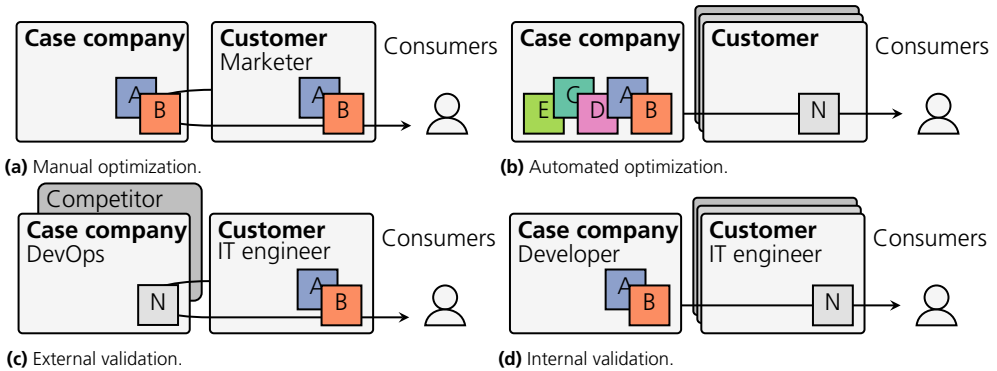


Figure 7: Experimentation scenarios at the case company in Paper II. The large rectangles depict the companies and the roles involved, arrows indicate the initiator of the experiments, the squares represent software components, with multiples of them—marked A and B—indicate variations of the software and a grey box with N indicate no change.

From the scenarios (see Figure 7), we see the following four general attributes. First, the *purpose* of the experiment is optimization or validation. In optimization experiments, the goal is to fine-tune software and interface parameters towards a given metric. For the validation purpose, the experiment is conducted to validate a change in software, such as a new software product or feature. Second, *automation* is whether the decision of the outcome is taken with manual analysis or automated with tools. Automated optimization experiments is the motivation for Papers VII and IV in the tool track, though automated validation experiments are also advocated for in the discussion in Paper IV, Section 7.1. Third, *control* of the software can be external or internal from the perspective of the experimenter. In an internal experiment the experimenter has full access to source code or the development organization. Finally, experiments have a *recurrence* of repeated or singular. The case company needed to repeat experiments for multiple customers.

In addition to optimization and validation, three further purposes of experimentation can be derived from related work [187, 264, 323] and from the interview material of the theory track. First, verification tests are used in software testing as a companion to validation tests [299]. Verification is an evaluation of whether software complies with requirements while validation is an assurance that the product meet the needs of stakeholders. Thus, experiments that are performed as part of a CICD pipeline by DevOps to survey a software release can be considered as *verification experiments*. The external validation scenario (c) qualifies, and verification should probably have been included in Paper II. Schermann et al. [264] refer to a similar concept as regression-oriented experimentation (compared to business-oriented experimentation). Second, experiments can be conducted with the explicit purpose of learning about users or the product. Linden [187] at Amazon, give an example of adding intentional delays to some of their queries to be able to quantify the value of future performance improvements in revenue. These *knowledge experiments* were not observed in the interview material of the theory track, and is likely a rare occurrence.

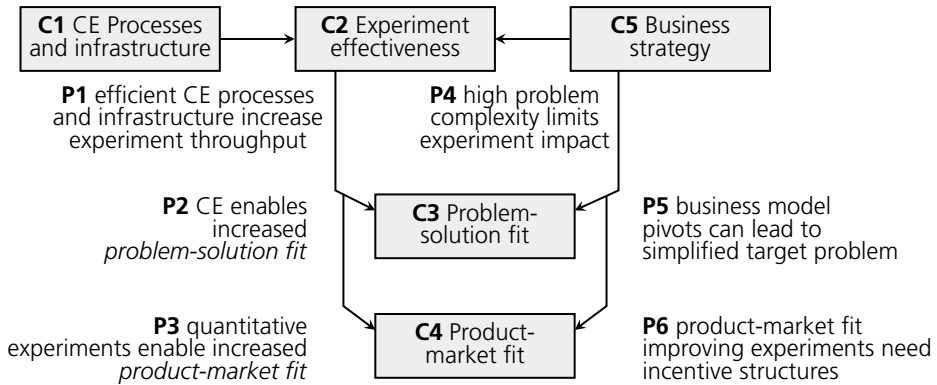


Figure 8: A theory of Factors Affecting Continuous Experimentation (FACE) which can be used to explain companies' ability to conduct CE. The theory constructs, **C1–C5**, are shown in rectangles and propositions, **P1–P6**, as arrows between constructs.

Finally, *calibration experiments* are conducted to verify the experimentation platform itself. For example, an A/A test with no software change can be performed a number of times to assess whether the false positive rate is acceptable [323]. These experiments were conducted at the case companies with experienced experimentation in Paper III.

4.3 Paper III: The FACE Theory

In Paper III, we wanted to investigate what factors influence how well companies can conduct their CE. This inquiry resulted in a SE theory which is summarized in Figure 8. We derive three factors from the theory: (1) investments in *data infrastructure*, (2) *user problem complexity*, and (3) *incentive structures* for experimentation. Furthermore, guidelines were derived to help practitioners to take action from the insights obtained from the theory. The guidelines are summarized in the DS abstract in Table 4. Another finding of the paper is how UX roles are involved with experimentation in SE, as described in Section 1.4.1.

The theory is based on data from a multi-case study of 12 companies and 27 interviewees. The companies have varying degrees of experimentation expertise and extent of experimentation in their organization. The company sizes range from small, with less than 50 employees, to huge multi-national corporations. The companies also differ regarding the type of business model, type of users, type of marketing strategy, etc. The goal of the study was to answer which of these differences that are the most relevant to a company's success with CE.

The theory is expressed as five constructs, C1–C5, and six propositions, P1–P6, which represent relations between the constructs (see Section 3.3.2). The constructs follow. C1 *CE processes and infrastructure* include the experimentation procedure in use at the company and the infrastructure for collecting data, deploying changes to software, and analysing results of experiments. The entire SE process is included in the construct; since CE encompasses the whole SE processes. C2 *experimentation effectiveness* is a factor of the organization's experimentation throughput and how much potential impact each experiment has. C3 *problem-solution fit* is how well the software product solves the users problem. C4 *product-market fit* is how well the product satisfies market needs. Both problem-solution fit and product-market fit is important, but product-market fit is what ultimately matter for businesses with products [96]. Finally, C5 *business strategy* is how a company is organized to create and deliver value to customers. That includes how it reaches customers, the revenue and licensing model, and more.

The propositions relate how the constructs affect experimentation in the following way. The factors are derived from the propositions that hinder or enable problem-solution fit or product-market fit and are denoted in italics. P1 infrastructure and process improves experimentation effectiveness. In particular, *data infrastructure* is a hurdle for CE. P2 experiments can improve either the problem-solution fit or P3 product-market fit. Many companies are unable to affect product-market fit due to P4 and P6. P4 the *user problem complexity* is the complexity of the problem the software solves for users and strongly limits experiment applicability. High complexity can limit the ability to make desired changes in the software or to quantify the user sessions. Following that, P5, pivots in the business model is necessary to simplify the problem complexity, experiments on their own are unlikely to succeed. Finally, P6 improving product-market fit needs the right *incentive structures*, e.g., in the form of having metrics from the sales process.

We derived guidelines from the FACE theory on how state-of-the-art CE is structured in *processes* and *infrastructure*. The companies with advanced *processes* use prioritization of both software development and experimentation with user-data. They conduct analysis before and after experiments, and make sure knowledge from experiments is shared in the organization. They utilize mixed-methods experimentation with both qualitative and quantitative methods. Their *infrastructure* is more mature too. Their data infrastructure cover all parts of the software and with more types of user data. Their experimentation platform can run experiments in parallel and supports segmentation of the results into groups of users. The overall CE competence is higher too, both in form of available roles and in knowledge of experimentation throughout the organization, such that all engineers can partake in CE.

Table 4: DS abstract for Paper III:
The FACE Theory for Factors at Play in Continuous Experimentation.

TR	To strive for state-of-the-art CE in adverse contexts apply the FACE derived guidelines.
Problem	Some companies are not able to conduct CE to the degree that they desire. Their experimentation might be lacking in, e.g., coverage of user groups or software features or in how relevant metrics are for users or business.
Solution	A theory of Factors Affecting Continuous Experimentation (FACE) is introduced to explain why company efficacy and utility of CE differ. Twofold practitioners guidelines are derived from FACE. First, a benchmark on how state-of-the-art experimentation is structured. Then, how companies can move towards more advanced CE based on the FACE factors affecting them.
Relevance	12 companies across domains and maturity of CE were involved in the study. Many interviewees expressed interest in improving their CE practice.
Rigor	The evidence chain from case study design [253], to theory construction [274], and to guidelines [278] all follow established guidelines.
Novelty	The FACE theory can explain factors for applicability or utility of CE.

Furthermore, we provide recommendations on how CE can be adapted and advanced at the companies who do not fulfill some of the three factors, as follows. (1) *Data infrastructure*. In the interviews it was said to be surprisingly technically easy to start with experimentation, but scaling to high throughput is hard. So we recommend companies to build processes and infrastructure gradually, such that the knowledge gained during practical experimentation can be used when scaling the infrastructure to high levels demand. (2) *User problem complexity*. Companies with high problem complexity should first of all seize all opportunities to pivot towards a simplified problem. The companies in the study with very high complexity are forced to rely on qualitative methods, which still enables them to work in a data-driven fashion. (3) *Incentive structures*. Companies without apparently suitable metrics need to put effort into finding the right goal to experiment on. An ideal goal will be good for both users and business value, so that either is not neglected when the goal is optimized. Furthermore, once a goal is in place, teams could be motivated to start with experimentation by improving the goal, since they are provided with the means to do so. Finally, be mindful of ethics when the goal is capable of being intentionally manipulated such that users suffer in favor of business value. To prevent this, goals should be set in collaboration between the teams and the rest of the organization such that the goals are scrutinized.

Table 5: DS abstract for Paper IV:

Data-driven Software Design with Constraint Oriented Multi-variate Bandit Optimization (COMBO).

TR	To <i>optimize, personalize, and adapt software to various usage</i> in user-intensive and potentially complex software apply <i>COMBO for continuous software optimization</i> .
Problem	Data-driven optimization of software parameters can drastically increase experimentation throughput. However, current approaches are limited to software variables with a flat hierarchy and few interactions or dependencies; which could be the reason for the lack of adoption of algorithmic optimization.
Solution	The toolkit Constraint-Oriented Multi-variate Bandit Optimization (COMBO) can model hierarchical variables with constraints in a domain-specific language; personalization is supported through the constraint system. The approach combines several approaches from different fields: machine learning, bandit optimization, and combinatorial optimization.
Relevance	The problem has been observed in industry [138, 208, II]. The research was conducted in an industrial collaboration to increase relevance.
Rigor	Two feature prototypes were implemented and were subjected to simulation experiments with user data. The research followed DS methodology [306].
Novelty	Several techniques are combined in a novel way.

4.4 Paper IV: The COMBO Toolkit

Design of software entail many decisions to be made in both visual design of the user interface and how the software is used in the user experience. There are also technical decisions that can inadvertently affect the user experience, for example, by making the interface unresponsive or by ranking things in a wrong order. Actually experimenting on all decisions in a software system is daunting, especially when interactions between functionality is considered. This is the motivation behind automating the decision making in experiments online in a production environment, with algorithms that can optimize the user experience based on user data. For this reason we introduce the Constraint Oriented Multi-variate Bandit Optimization (COMBO) toolkit, the research is summarized in Table 5.

The overall idea of COMBO is to parametrize multiple software variables with a range of possible values in a search space and have a decision algorithm optimize the selection of variable configurations for users. The algorithms in COMBO build on the multi-armed bandit problem in statistics (see Section 1.5), the application of which is referred to as *bandit optimization*. Figure 9 give an overview of the setting. When a *new user* arrives to the *software system*, they receive a *configuration* adapted to their personal or usage *context*. They use the configuration in the software system for some time and then a *machine learning* algorithm is *updated* based on the configuration and some measure of the user experience—referred to as a *reward*. An *optimizer policy* is tasked with optimizing the overall rewards over time based on the learned *model parameters* by experimenting with users’ configurations.

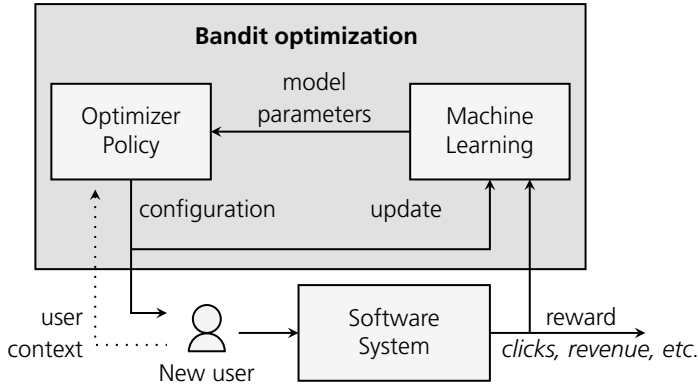


Figure 9: Bandit optimization setting summary for handling automated experimentation.

Automated experimentation is one of the scenarios found in Paper II and was found in Paper I to be used in *industry* in at least two more instances [138, 208]. These instances were all in e-commerce, a domain in which experimentation is widespread, as shown in Paper V. The purpose of the three uses are as follows. (1) To optimize the user experience for different users’ needs (i.e. *personalization*) with multi-variate bandit optimization [138]. (2) To optimize user interfaces with many variables in e-commerce using multi-armed bandits and genetic algorithms [208]. Finally, (3) to tune parameters of a recommender system [41] with multi-armed bandits (see Figure 7b). The problems that the solution is addressing in these examples are all of low technical complexity which limits the applicability of the solution designs.

Modern software is increasingly technically complex and much of software engineering practice is about handling this complexity [17]. Structuring software with modularity and hierarchical abstractions is an ubiquitous approach to dealing with the complexity [192]. The modules can be studied and developed independently from each other and the hierarchical abstractions can hide implementation details such that the modules can be more easily composed in an overall architecture. It follows then, that to be able to optimize technically complex software, these hierarchical abstractions must be able to be expressed in the search space of the automated experimentation. This is not possible in the previous research on automated experimentation [138, 208] and this is the main problem that the COMBO toolkit addresses.

The search space in COMBO is specified in *feature models* [158], i.e., an approach to formally model arbitrary abstraction hierarchies in SE research. It has been used to describe the variability of software [24, 70] in software product lines [292]. The point is to express constraints between features and to formally verify that the constraints hold under various conditions corresponding to actual realizations of the software [21]. Feature models has also been suggested to be useful to model experimentation by Cámara and Kobsa [48].

The technical and algorithmic contributions of Paper IV is fourfold. First, the multi-variate bandit optimization setting and its solution is introduced to the general SE audience, see Figure 9. The setting is similar to Bayesian black-box optimization [142, 275] but more suited to combinatorial variables with a low signal-to-noise ratio and to have the optimization be performed in runtime. Second, we showed how to add explicit constraints to bandit optimization using combinatorial optimization techniques. Third, we suggested how to adapt popular online machine learning algorithms to handle bandit optimization with constraints: VFDTs [84], random forests [110], generalized linear models [138], and neural networks [241]. Finally, we showed how these ideas can be applied in practice to automated experimentation by implementing an open source toolkit. The toolkit consists of an embedded domain-specific language for specifying the search space with constraints and several algorithms for machine learning and combinatorial constraint solvers and optimizers.

The research in Paper IV was *validated* by implementing prototype models that describe the variability of two features at the case company: an auto-complete search widget and a top- k category recommender system. We used simulation experiments for evaluating the performance of the algorithms on the feature prototypes using a combination of real and generated user data. The performance was measured on both computing times and reward performance.

Finally, the paper contain an analysis of the implications of when companies embrace toolkits like COMBO. The CE process can then move from experimentation to a *continuous optimization* (CO) approach. The CO practice extends CE by giving developers the option to use an optimization algorithm at low cost. Thereby enabling large parts of the software to be optimized and personalized. Developers will continuously engage with the optimization algorithm by, e.g., adding or removing variables from the search space as they learn more about the usage domain. This means using an optimization algorithm for much more than an isolated component of a product.

The suggested CO process can be structured in four steps. First, investigate and prioritize the variability of a feature by user experience research methods, data mining, or prototype experiments. Second, build a model of the variables in the search space and add constraints to prune invalid configurations. Third, tune the algorithmic performance in offline simulations with historic or generated user data and formally verify the model using feature model variability management techniques. Finally, validate the solution in a controlled experiment in a production environment and restart the process.

Table 6: Synthesized DS abstract for papers I-IV.

TR	To achieve <i>problem-solution and product-market fit</i> in <i>user-intensive software</i> use the <i>FACE guidelines</i> to reach <i>CE</i> and the <i>COMBO toolkit</i> for reaching <i>CO</i> .
-----------	---

Problem	Decisions on what software features to build and deploy are not based on evidence on what changes best benefit users. This can lead to software that does not solve the users' problems or that has no market. Some companies have adopted experimentation to increase problem-solution fit. However, not all companies are able to reach experiment-driven software development with CE. Furthermore, even fewer companies can target product-market fit with experiments. For those that do, they might not be able to keep up with the degree of optimization experiments they desire.
Solution	Companies can gauge their experimentation based on the state-of-the-art guidelines presented in FACE and obtain a deeper understanding on how their CE is impacted by the factors of FACE. Furthermore, COMBO can further elevate experimentation to optimize, personalize, and adapt software to various usage. COMBO functions by automating the decision making with machine learning based constraint-oriented bandit optimization. By fully adopting an optimization approach, companies can move from CE to CO, where the optimization problem is continuously updated with new variables and constraints.

Relevance	The problem was derived from related work found in the systematic literature review of Paper I and in the multi-case study of Paper III. Experimentation and optimization is done at many software companies of different sizes, business strategies, and sectors; with varying degree of success. The case companies in Paper III cover a wide range of these aspects.
Rigor	The solutions are validated with real companies and CE is observed at 12 companies that all express interest at improving their experimentation. The empirical research methods include case studies, and simulation experiments based on real world data. Guidelines were adhered to for all steps of the research where applicable.
Novelty	The FACE theory and the COMBO toolkit for CO are novel contributions.

We surmise that the introduction of CO will improve software in two ways compared to CE. First, many more variables can be searched for and co-optimized. This will improve software because the search process is more likely to find well performing configurations due to considering more options and more crucially it considers interactions between variables. Second, software can either be personalized to suit individual needs or functionality that is re-used can be adapted to different usage contexts.

5 Synthesis

This section contains an analysis of how the research goals are addressed by the papers in the project. The combined DS abstract is presented in Table 6 which summarizes the coherent contribution of the thesis.

5.1 RG1: Map and synthesize the currently published knowledge on CE

The research goal to map and synthesize the existing research has been addressed directly by the papers [V, I] in the literature track. The published research was synthesized with thematic analysis into the categories as presented in Figure 6 on page 20. It is clear that the research has been focused on CE challenges and solutions with comparatively few conceptual contributions—which indicates an emerging field.

The results from Papers V and I also show that there are many industrial authors in CE research, publishing experience reports and proposed solutions that address specific problems at companies. The large web companies, such as Google [286], Microsoft [172], and Facebook [107], are over-represented in the publications. However, there are several case studies by software engineering researchers on all sorts of companies interested in CE [318, 242]. Together, this shows that the research is very relevant to industry.

Many of the solution contributions are also solution instantiations (see Figure 4 on page 12) without any solution abstraction, where a solution instance is put into practice without an analysis of generalized knowledge that can be obtained from it and how it can be applied to other contexts. This is presumably a consequence of the industrial dominance in the field, due to industrial authors' rational focus on solving the specific problems that face their company. As such, validating and abstracting solutions represents a research opportunity for academic researchers, which has also been done recently to some extent [10, IV]. Solution abstraction has been given attention to in the COMBO paper [IV] by combining two of the solution types found in the literature.

5.2 RG2: Understand current industry practice and state-of-the-art of CE

The papers in the theory track [II, VI, III] show that CE practice is adopted in industry, though irregularly. Some companies are able to conduct experimentation with a sophistication and extent that others cannot. The reason for this discrepancy can be summarized in the three factors derived from the FACE theory [III]: (1) data infrastructure investments, (2) user problem complexity, and (3) incentive structures. For example, a company with insufficient infrastructure might not be able to experiment with relevant data or a company with high user problem complexity might not be able to quantify their user experience. The FACE theory can be used to guide research by informing future studies, the factors can be used by practitioners to understand their practice, and the guidelines show how to move towards state-of-the-art practice.

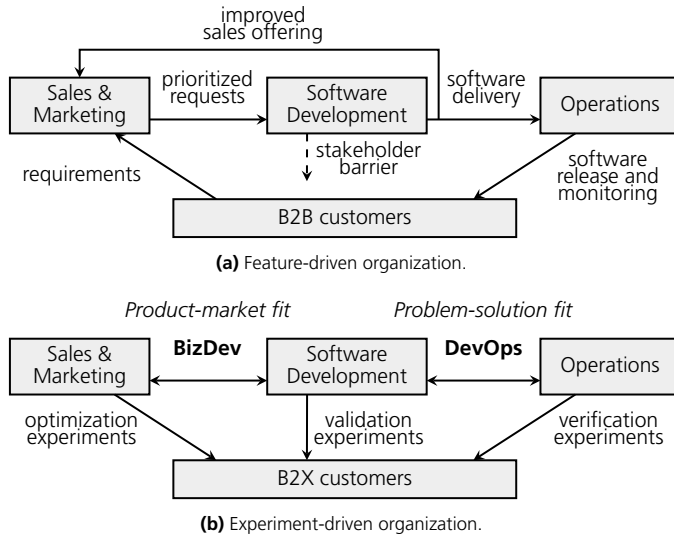


Figure 10: Illustration of how two archetypes of software-intensive organizations conduct software engineering projects with involvement from the three customer-facing departments (adapted and extended from Paper VI).

Figure 10 summarizes the implications of the FACE theory. Based on how the case companies in the multi-case study [III] use CE, we see two main archetypes of software development organizations: (a) *feature-driven* organizations and (b) *experiment-driven* organizations. These organizations interface with customers in very different ways as explained below. While both organization types may have user-intensive software, the feature-driven organizations generally have business-to-business (B2B) customers, because the process does not scale to individual users. The experiment-driven organizations can have both business-to-business and business-to-consumer (B2X) customers. Other terms have been suggested for similar concepts with different points of view: sales-led versus product-led growth [18, VI], market-driven versus customer-driven business models [72], and requirements-driven versus data-driven processes [36]. These dichotomies represent an idealized view, e.g., there are organizations in-between and some feature-driven organizations may make heavy use of experiments.

The overall software engineering process of the *feature-driven* organizations is focused on fulfilling customer feature requests. Customers pose requirements in the form of feature requests to the sales & marketing department. The customers' feature requests are prioritized along with feature ideas from the sales & marketing department, based on what features would benefit the most important customers. The software development department then implements and makes a software release, which is delivered to the operations department and reported to the sales & marketing department; such that the delivered release fulfills the customer requirements. Experiments can be used at these organizations too, to increase

the quality of the release with verification experiments. Note that the software development department is far detached from customers in Figure 10a. In Paper VI, this observation is referred to as the *stakeholder barrier* because developers find it easier to go the first stakeholder they have access to, which is the sales & marketing department instead of customers.

In an *experiment-driven* organization, the development process is continuous, and the three departments in Figure 10a are integrated and draw mutual benefit from each other by making software experiments; in accordance with the BizDev and DevOps concepts (see Section 1.2). BizDev and DevOps have different goals for their collaboration, respectively, to optimize product-market fit and problem-solution fit (see Section 4.3). In contrast to the feature-driven organizations, the customers here are passive providers of feedback through measurements on their usage of the product. The three departments are all able to conduct experiments as in the scenarios of Paper II. The sales & marketing department can use experimentation tools to *optimize* the sales funnel, the software development department can *validate* that software changes work as intended, and the operations department can *verify* that changes do not break or degrade any of the continuous software releases.

The FACE theory implies that the reason why BizDev is not in place at many feature-driven companies is that the software development and sales & marketing departments have differing incentives for improving the software product or sales offering, respectively; building the thing versus selling the thing is synergistic but not the same. Incentive alignment for DevOps is likely easier to achieve than for BizDev; several sources cite aligned incentives or goals as the enabler of DevOps [145; 147, Chapter 5; 228]. Also, DevOps can clearly exist also in feature-driven organizations and indeed there is an overlap between DevOps and feature-driven organizations in the interview material of the theory track. However, an experiment-driven organization is more likely to have a larger user base (see Paper III, Section 5.5.3) with higher demands on infrastructure and thus will have a greater need of DevOps principles and therefore also a higher degree of CE.

5.3 RG3: Support practitioners in analysing and improving their CE practices

The final research goal is addressed by both the theory track and the tool track. The FACE theory [III] can support practitioners in analysing their ability to perform CE and what factors influence this. The guidelines derived from the theory describe what state-of-the-art CE consist of and how companies can improve their experimentation based on their specific company context. The COMBO toolkit provides practitioners with the possibility to personalize and adapt their software to different usage. However, not all companies can use the toolkit. Most likely all three factors as described by the FACE theory must be fulfilled, for this to be possible. In particular, applying the techniques places high demands on data infrastructure and the software usage must be able to be expressed as a single unambiguous metric that can be optimized for.

Feature-driven organizations that do A/B tests on selected new developments may want to increase their degree of experimentation to correspond to a more continuous use of the practice. In that case, the FACE theory guidelines can help guide their transition to CE. The most straightforward (and challenging) option is to pivot the business model such that the incentives between users and the company are aligned, i.e., by tying the customer revenue stream to product usage. In this way the product-market fit can be targeted in experiments by using sales metrics. This will in turn also align the incentives between the sales & marketing and software development departments. As an example of this in practice, consider selling software with subscriptions, the connection between whether users start or stop the subscriptions can then be used in experiments. In contrast, software that is sold upfront cannot take advantage of sales figures in experiments in this way. Thus, the experimentation from the software development will be limited to increasing problem-solution fit with behavior data only. As for the sales & marketing department, they will have more incentives to experiment on marketing material than the product because it will have a more direct impact on sales. Consequently, product-market fit will not be reached through experiments for feature-driven organizations.

Of the three experiment types in Figure 10b, the optimization experiment type seems like the least understood; both in SE literature due to being partly a marketing or UX activity, and in SE practice due to being of a lesser priority than validity or verification experimentation—because problem-solution fit should be reached before product-market fit [201]. Also, this weak understanding might be as a consequence of that the concept (or practice) of BizDev is less established than DevOps. Some of the larger companies investigated in the theory track do however make use of optimization experiments and consider these experiments as an important part of their daily work. Two of the case companies have growth engineering or marketing departments that are responsible for experimenting to increase product-market fit (and problem-solution fit to a lesser degree). These departments make changes to the software directly and employ data scientist and software developers. Albeit, many of the sales & marketing departments at the case companies perform optimization experiments on marketing material, in isolation and with separate infrastructure.

Very large companies, like Google and Facebook, have the manpower to perform optimization experiments continuously (i.e. CO) to a very high degree and this practice is spread throughout the organization. They have presumably reached CO without broad optimization algorithm tool support. The infamous experiment at Google to find which of 41 shades of blue to use for clickable links is one example of their optimization [141]. The example is reported to have increased yearly revenue with 200 million dollars at the time [229]. Another example, also from Google, explains how a software engineer was involved in a debate about how many pixels wide a border should be, when pressed to settle the discussion with an A/B test they decided it was the tipping point that made them quit Google [38].

The suggestion in Paper IV is to instead support CO through the use of algorithms that optimize the assignment of users to configurations. CO needs support like what is suggested in Paper IV for allowing easy specification of experiments so that the threshold to use tool support is low. In this way, the debate in the example above could be avoided by specifying a range of pixels to use as a border and have the system handle the rest; though COMBO would need to be further integrated to support such easy development. Adoption of a toolkit like COMBO would enable a CO approach at a lower manpower cost. Algorithms for CO are in place at some companies, for narrow usages (see Section 1.5). However, widespread adoption of CO likely requires that toolkits such as COMBO reach production quality. There are advantages to CO beyond increased experimentation capacity. Namely, for personalization and adaption of the software to different contexts, e.g., device type, country markets, etc.

6 Discussion

The methodology and ethics of any research effort should be critically assessed. This section contain the following analysis of the research: ethical considerations of CE, ethical considerations of the research project, threats to validity, and remaining future work. Engström et al. [98] argue that DS research should be judged on relevance, rigor, and novelty aspects, which are described for each paper in the DS abstracts in Section 4 and for the combined thesis in Section 5. The rigor aspect is further expanded on in this section with the threats to validity.

6.1 Ethical considerations for the practice of CE

There are many ethical aspects to consider when *conducting experiments with human participants*. For example, to what degree should users' data privacy be guaranteed, and whether users should need to actively consent to be part of experiments. Ethical considerations for conducting CE on real users are called out as a topic for further research in Paper I.

A study by Kramer et al. [179] from 2014 is an interesting case that illustrates the ethics issues, causing concern in the research community at large and in news outlets. The researchers conducted an experiment on Facebook to study the effect that posts on their social media website with negative sentiment had on other users' mood. The research was conducted with good intentions in order to learn more about the psychological effect that social media has on users (the contagion effect in the study was significant but small). However, the research could potentially cause harm to individuals. Benbunan-Fich [22] argues

that this type of subtle behavior-manipulating experimentation should be viewed differently from the type of product-improving experimentation which is the topic of this thesis. Though, it is conducted with the same technology so the distinction can be hard to make for legislators.

An external ethical review board was not consulted to approve the experiment at Facebook [207]. The authors did so retrospectively after the backlash, and it was found to be unnecessary due to the research being conducted by a company for internal reasons; that the research was publicized seem to be a separate matter. According to the editorial expression of concern [296], another issue is that the participants in the experiment on Facebook were not asked whether they wanted to participate in the research (informed consent) and were thus not given a chance to opt out. Also, only after four months of publication did Facebook update their terms of service to mention that experimentation was conducted [139]. Now, users of Facebook agree to participate in any testing, research, and data analysis through the terms of service. This implicit consent also appears to be industry standard practice [172], both at the time when Facebook conducted their experiment [179] and now.

Whether explicit consent should be required to be given by users to be included in a product-improving experiment is up for debate. Whether users should even be informed of each experiment that they are part of is also debatable. Yaman et al. [319] studied experimentation consent from the perspective of different roles in a SE organization. They found that developers are inclined to withhold information to users if the result depend on it. If users are aware of the experiment they might modify their behavior which might impose unwanted cognitive biases [211].

Another ethical consideration is whether experiments can be used to drive discrimination. Jian et al. [150] showed how experimentation is used in actual practice to intentionally discriminate against certain demographics, i.e., by increasing prices. However, Niculescu et al. [216] showed how it can also be used to decrease discrimination by using it as a target metric in experimentation. Hence, whether or not experimentation is a driver for or against discrimination depends on the experimenter and the specifics of the experiment.

Data privacy issues are less of a problem since they are possible to address as part of the data infrastructure and there is clear regulation with the General Data Protection Regulation act (GDPR) [65, 297]. Standard controlled experiments do not need to store any information about individuals beyond a user or session token of some sort [172] (a token is a long random number used by websites to remember a user's history without knowing who they are). However, when experimenters segment data in order to get results for various demographics or clusters of users then data about each user need to be analyzed, in a

GDPR-compliant way. Care must be taken since there is always a risk that data can accidentally identify individuals. For example, in one of the cases of Paper III, the geographical location of each user token was stored, and in order to be GDPR-compliant they needed to add noise to the data to anonymize it.

6.2 Ethical considerations of the research project

The research in the thesis includes human participants at companies and there are risks involved for both the interviewees and the companies with how interview data is handled and reported. For example, risks that the interviewees reveal sensitive information about company internals or that the interviewees can be identified and get in trouble with management for statements made during their interview. The following procedure for handling the risks were adhered to as recommended in the guidelines for case studies by Runeson et al. [253]. Interviewees were informed about the interview procedure (see Paper III, Section A) and consent was obtained from them about the following: that the interview is recorded, that they are free to withdraw from the study at any time until publication (with their interview data), and what the purpose of the study is. The original data from the interviews are stored securely by the main author and not shared. Anonymized transcripts were shared with co-authors for validation purposes. The descriptions of case companies and interviewees are reported as general as possible while still retaining the information relevant to the research, thereby minimizing the risk of identification.

6.3 Threats to validity

The relevant validity and reliability threats are discussed here, for each research track and lastly for the combined thesis. *Validity* deals with how accurate the results are and *reliability* relates to how reproducible the results are. There are multiple types of validity threats and different types are judged to be more important for different steps along the DS cycle (see Section 3.1). In Figure 11, a mapping is provided from threat types to the DS knowledge creating activities. For research bridging between the problem-solution domains, all three of *external validity*, *internal validity*, and *reliability* are important to ensure that the solution design is suitable to multiple problem constructs and that the empirical evaluation of problem-solution instances is trustworthy and repeatable, respectively. *Construct validity* is crucial for theorizing practice, by ensuring that the conceptualized problem or abstracted solution instances are correctly represented in the resulting constructs. Finally, when putting theory into practice, *instantiation validity* is needed to ensure a design construct fits and solves a specific problem instance. These validity types are standard in SE research [91, 253], except for instantiation validity which is an addition specific to design science [191].

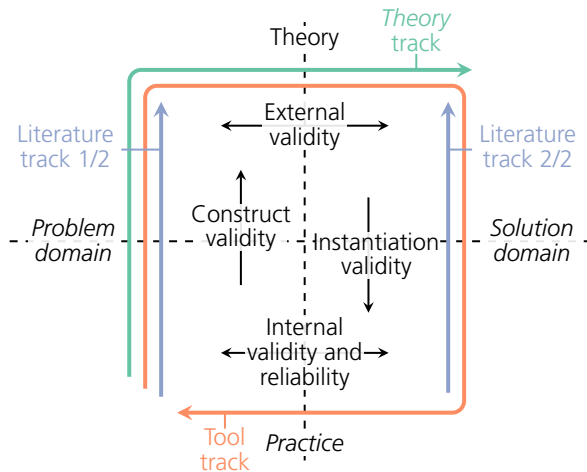


Figure 11: Threats to validity and reliability mapped to the design science cycle in Figure 4 on page 12. The paths of the three research tracks are shown in the outer arrows.

6.3.1 Literature track validity

Construct validity is the most important validity aspect for the literature track, though internal validity and reliability are also crucial. The search, selection, and analysis process was conducted in parallel by two authors and combined afterwards by analyzing conflicting choices, this increased *internal validity* and highlighted the *reliability* of the research process. Furthermore, the threat to *external validity* is minor due to being an aggregation of an industry relevant field; hence the research is inherently transferable to CE research and practice.

Construct validity is addressed in the three following ways. First, the process used guidelines covering all parts of the SLR process from search and selection [169] to analysis [68, 144]. Second, the resulting themes have low levels of abstraction to limit the risks of misinterpreting the primary research. Finally, coverage of the body of research is critical for construct validity in literature studies, e.g., such that the synthesis of available solutions is complete. The extensive search process ensured sufficient coverage in the included papers.

6.3.2 Theory track validity

In the theory track, the primary concerns are construct validity in the conceptualizations and external validity of the guidelines such that they can describe and improve CE practice for other case companies. Both Papers II and III in the track make use of guidelines for case study methodology [253] and thematic analysis [68] in a systematic way to increase

construct validity. Paper III also uses the systematic theory building process by Sjøberg et al. [274]. The threat to construct validity was further limited in the case selection by striving to include a range of cases from companies with extensive usage of CE to those with next to no CE usage.

Furthermore, *external validity* was considered by broad case selection, such that the likelihood was increased of that the constructs and propositions are generalizable to other companies. External validity is further supported by thoroughly describing the context of each included case company, such that practitioners can judge the applicability of the theory in relation to their context. In Paper II, the scenarios were derived in a company that served multiple customers, hence the external validity is higher than what would be expected for a single case. However, verification experiments were not included in the paper (see Section 4.2) due to insufficient coverage of the concept at the time.

6.3.3 Tool track validity

The tool track consists of a full DS cycle and thus all aspects of validity are important. *Construct validity* is strengthened by following the design science methodology proposed by Wieringa et al. [306] so that the abstractions proposed in the toolkit were made to support specific use cases. *External validity* is addressed by disclosing the limitations of COMBO and the intended setting. The use of the toolkit was motivated by related industrial work [138, 209] so the research is relevant. *Instantiation validity* is crucial to have the toolkit be applicable for practitioners. The cost of instantiating the toolkit is substantial, since it needs close integration with a production software system, hence limiting the ability to academically validate the toolkit. However, the design decisions of COMBO were described and the software is open sourced to increase instantiation validity. Also, the solution is abstracted to a construct such that conceptual aspects of the contribution in Paper IV can be used without relying on instantiating the whole toolkit. *Internal validity* is increased by using data from real users in a systematic way. Finally, *reliability* in the form of repeatability of the validation is improved by providing source code and replication instructions.

6.3.4 Thesis validity

In summary, the overall thesis validity of the research is analyzed as follows. *Construct validity* is addressed by adhering to established research guidelines for all studies [169, 253, 274, 306]. *External validity* is improved by having started the research project with a literature study that included descriptions of CE at many companies and by including multiple case companies in the research. *Instantiation validity* is increased in COMBO by abstracting the solution such that the contribution does not rely on instantiation. *Internal validity* is

strengthened by the use of multiple types of contributions such that CE is studied from multiple angles, i.e., triangulated. Finally, *reliability* is increased through theoretical repeatability by being transparent in making source code, code books, and interview guides available.

6.4 Future work

There is still remaining research to address in future work. In the *literature track*, the research contributions include several items of future work for the CE field as a whole. Since the research field includes many industry authors, the overall recommendation for future work is to synthesize and evaluate existing models or solutions. Several of the concrete suggestions for future research are still unaddressed, in particular, a process model derived and validated from more cases would be a valuable contribution. The interview material from the *theory track* includes several questions about processes and could be used for this purpose. In the theory track, extending Paper II to a full taxonomy of attributes of experimentation would be a valuable reference for both practitioners and academic researchers. Also, further validation of the FACE theory in Paper III is called for, by comparing with CE practice. In the *tool track*, further empirical validation of the COMBO toolkit is also needed, preferably with an industrial case where the toolkit optimizes the user experience of real users.

7 Conclusions

The overall goal of the thesis is to *describe current practice and support continuous experimentation (CE) in industry*. The research was conducted with empirical methods and observations of real world phenomena using a design science frame. The research goal has been addressed with the following four papers, included in the thesis:

Paper I is a systematic literature review that gives an overview of the previous research within the field that has been conducted with both a researcher and practitioner point of view in mind. Many of the primary papers included in the review have industrial authors that publish problem-solution instances or challenges they faced in their practice. However, it is unknown how many of these observations are transferable to other companies' contexts. Thus, the field is industry relevant but could benefit from further research with industry-academia bridging collaborations.

In Paper II, an initial case study was conducted at an e-commerce company to give a more nuanced view on how CE is applied in industry, which resulted in four scenarios of experimentation. Primarily, the results highlight that experiments are either done for optimization purposes or to validate new feature developments. The scenarios also show that

experimentation is conducted in all three of the sales & marketing, software development, and operations departments; albeit with differing tools and goals. The tools for optimization at the sales & marketing department are particularly extensive while the tools for the software development department's validation experiments are flexible.

The initial case study was followed by Paper III, with 11 additional case companies and expanded theoretical scope. This resulted in a theory on the underlying factors that explain the degree to which companies can expect to derive utility from CE. The identified factors include CE processes and infrastructure, how complex the problem is that the software solves for users, and the type of business model used to monetize the software. Empirically-based guidelines on how companies with state-of-the-art practice structure their CE and how companies can adapt their CE to their factors are provided. Companies with successful CE integrate their experimentation as part of BizDev and DevOps.

Finally, in Paper IV, a toolkit for conducting automated experimentation is presented where developers with limited data science knowledge can optimize software in a larger scale and to a greater end result. The toolkit includes a domain-specific language for specifying what variables to optimize on and how they interact with constraints. The constraint system enables personalization support of the software to users' context. The paper includes an analysis of practical considerations for how a process of *continuous optimization* that extends CE can be implemented at companies.

In conclusion:

- Companies gain different degrees of utility from continuous experimentation which the FACE theory illustrates. The guidelines derived from the theory can help companies reach state-of-the-art experimentation.
- The COMBO toolkit supports practitioners wanting to elevate their experimentation with an optimization approach to improve and personalize software to users' contexts.

Included Papers

Paper I

Controlled Experimentation in Continuous Experimentation: Knowledge and Challenges

Florian Auer Rasmus Ros Lukas Kaltenbrunner Per Runeson
Michael Felderer

Abstract

Context Continuous experimentation and A/B testing is an established industry practice that has been researched for more than 10 years. Our aim is to synthesize the conducted research.

Objective We wanted to find the core constituents of a framework for continuous experimentation and the solutions that are applied within the field. Finally, we were interested in the challenges and benefits reported of continuous experimentation.

Method We applied forward snowballing on a known set of papers and identified a total of 128 relevant papers. Based on this set of papers we performed qualitative narrative and thematic synthesis to answer the research questions.

Results The framework constituents for continuous experimentation include experimentation processes as well as supportive technical and organizational infrastructure. The solutions found in the literature were synthesized to nine themes, e.g., experiment design, automated experiments, or metric specification. Concerning the challenges of continuous experimentation, the analysis identified cultural, organizational, business, technical, statistical, ethical, and domain-specific challenges. Further, the study concludes that the benefits of experimentation are mostly implicit in the studies.

Conclusions The research on continuous experimentation has yielded a large body of knowledge on experimentation. The synthesis of published research presented within include recommended infrastructure and experimentation process models, guidelines to mitigate the identified challenges, and what problems the various published solutions solve.

Keywords Continuous experimentation · Online controlled experiments · A/B testing · Systematic literature review

1 Introduction

Deciding which feature to build is a difficult problem for software development organizations. The effect of an idea and its return-on-investment might not be clear before its launch. Moreover, the evaluation of an idea might be expensive. Thus, decisions are based on experience or the opinion of the highest paid person [171]. Similarly difficult is the assessment of technical changes on products. It can be difficult to predict the effect of a change on software quality, as evidenced by the extensive research on, e.g., defect prediction [109, 298] or software reliability estimation [244]. Moreover, there are cases in which it is not feasible to test for all necessary conditions, e.g., for all relevant software and hardware combinations.

Continuous experimentation (CE) addresses these problems. It provides a method to derive information about the effect of a change by comparing different variants of the product to the unmodified product (i.e. A/B testing). This is done by exposing different users to different product variants and collecting data about their behavior on the individual variants. Thereafter, the gathered information allows making data-driven decisions and thereby reducing the amount of guesswork in the decision making.

In 2007, Kohavi et al. [171] published an experience report on experimentation at Microsoft and provided guidelines on how to conduct *randomized controlled experiments* in software engineering practice. It is the seminal paper about continuous experimentation and thus represents the start of the academic discussion on the topic. Three years later, a talk from the Etsy engineer Dan McKinley [203] gained momentum in the discussion. In the talk, the term *continuous experimentation* was coined to describe their experimentation practices. Other large organizations, like Facebook [107] and Netflix [125], which adopted data-driven decision making [174], shared their experiences [30] and lessons learned [176] about experimentation over the years with the research community. In addition, researchers from industry as well as academia developed methods, models and optimizations of techniques that advanced the knowledge on experimentation.

After more than ten years of research, numerous work has been published in the field of continuous experimentation, including work on problems like the definition of an experimentation process [106], how to build infrastructure for large-scale experimentation [130], how to select or develop metrics [193], or the considerations necessary for various specific application domains [94]. The purpose of this systematic literature review is threefold. First, to synthesize the models suggested by the research community to find characteristics of an essential framework for experimentation. This framework can be used by practitioners to identify elements in their experimentation framework. Second, to synthesize the various

technical solutions that have been applied. In this inquiry, we also include to what degree the solutions are validated. Finally, to summarize and categorize the challenges and benefits with continuous experimentation. Based on this the following four research questions are addressed in this work:

- **RQ1** What are the core constituents of a CE framework?
- **RQ2** What technical solutions are applied in what phase within CE?
- **RQ3** What are the challenges with CE?
- **RQ4** What are the benefits with CE?

The research method of this study is based on two independently conducted mapping studies [9, 249]. We extended and validated the studies by cross-examining the included studies. Thereafter, we applied qualitative narrative and thematic synthesis on the resulting set of papers to answer the research questions.

In the following Section 2, an overview of continuous experimentation and related software practices is given. Next, Section 3 describes the research method applied and Section 4 presents the results of the research. In Section 5, the findings are discussed. Finally, Section 6 summarizes the research.

2 Background

In this section we present an overview of continuous experimentation and related continuous software engineering practices. Further, we summarize our two previously published mapping studies. For the novice reader, we recommend Fagerholm et al.'s descriptive model of continuous experimentation [106], or Kohavi et al.'s tutorial on controlled experiments [172], which is a more hands on introduction for continuous experimentation.

2.1 Continuous software engineering

In their seminal paper on controlled experiments on the web from 2007, Kohavi et al. [171] explain how the ability to continuously release new software to users is crucial for efficient and continuous experimentation, which is now known as continuous delivery and continuous deployment. Together with continuous integration, these are the three software engineering practices that allow software companies to release software to users rapidly and reliably [270] and are fundamental requirements for continuous experimentation.

Continuous integration entails automatically merging and integrating software from multiple developers. This includes testing and building an artifact, often multiple times per day. *Continuous delivery* is the process by which software is ensured to be always in a state to be ready to be deployed to production. Successful implementation of continuous integration and delivery should join the incentives of development and operations teams, such that developers can release often and operations get access to powerful tools. This has introduced the DevOps [92] role (and concept) in software engineering with responsibility for numerous activities: testing, delivery, maintenance, etc. Finally, with *continuous deployment*, the software changes that successfully make it through continuous integration and continuous delivery can be deployed automatically or with minimal human intervention. Continuous deployment facilitates collection of user feedback through faster release cycles [99, 318]. With faster release cycles comes the ability to release smaller changes, the smaller the changes are the easier it becomes to trace feedback to specific changes.

Fitzgerald and Stol [113] describe many more continuous practices that encompass not only development and operations, but also business strategy; among them continuous innovation and continuous experimentation. Experiments are means to tie development, quality assurance, and business together, because experiments provide a causal link between software development, software testing, and actual business value. Holmström Olsson et al. [223] describe how “R&D as an experiment system” is the final step in a process that moves through the continuous practices.

2.2 Continuous experimentation

The process of conducting experiments in a cycle is called *continuous experimentation*. The reasoning is that the results of an experiment often begets further inquires. Whether the original hypothesis was right or wrong, the experimenter learns something either way. This learning can lead to a new hypothesis which is subject to a new experiment. This idea of iterative improvement is known since long from the engineering cycle or from iterative process improvements, as explained in the models Plan-Do-Check-Act [73] or quality process improvement paradigm (QIP) [19]. The term “continuous experimentation” as used by software engineering researchers refers to a holistic approach [113] which spans a whole organization. It considers the entirety of the software life-cycle, from business strategy and planning over development to operations.

Some authors have included many methods of gathering feedback in continuous experimentation [106, 188], including qualitative methods and data mining. These methods are not the focus of this work, though they are also valuable forms of feedback [37, 318]. For example, qualitative focus groups in person with selected users can be used early in development on sketches or early prototypes. The human-computer interaction research field has studied this extensively—recently under the name of *user experience research*—and it has

also been the subject of software engineering literature reviews in combination with agile development [156, 257]. In contrast to the qualitative methods, a controlled experiment requires a completed feature before it can be conducted. It is focused on quantitative data, thus cannot easily answer questions on the rationale behind the results, as qualitative methods can. As such, these methods compliment each other, but they are different in terms of methodology, infrastructure, and process. We discuss the qualitative methods through the lens of controlled experimentation in Section 4.2.9.

A *randomized controlled experiment* (or A/B test, bucket test, or split test) is a test of an idea or a *hypothesis* in which variables are systematically changed to isolate the effects. Because the outcome of an experiment is non-deterministic, the experiment is repeated with multiple *subjects*. Each subject is randomly assigned to some of the variable settings. The goal of the experiment is to investigate whether changes in the variables have a causal effect on some output value, usually in order to optimize it. In statistical terminology, the variable that is manipulated is called the independent variable and the output value is called the dependent variable. The effect that changing the independent variables has on the dependent variable can be expressed with a statistical *hypothesis test*. A significance test involves calculating a *p*-value¹ and the hypothesis is validated if the *p*-value is below a given *confidence level*, often 95%. In addition, properly conducting a controlled experiment requires a power calculation² to decide experiment duration.

In software engineering, a controlled experiment is often used to validate a new product feature, in that case the independent variable is whether a previous baseline feature or the new feature should be used. These are sometimes called control and test group, or the A and B group, in which case the *experiment design* is called an A/B test. In an A/B test, only one variable is changed; other experiment designs are possible [112, 172] but rarely used [249]. To optimize software configuration settings is another use of controlled experiments in software engineering [184]. The dependent variable of the experiment is some measurable metric, designed with input from some business or customer needs. If there are multiple metrics involved with the experiment, then an overall evaluation criteria (OEC) [252] can be used, which is the most important metric for deciding on the outcome of the experiment. The subjects of the experiments are usually users, that is, each user provides one or more data points. In some cases the subjects are hardware or software parameters, for example, when testing optimal compiler settings.

¹*T*-test is often used to compare whether the mean of two groups are equal, based on the *t*-score $t = \frac{\bar{x}_1 - \bar{x}_2}{s/\sqrt{n}}$, where \bar{x} is mean, s is the standard deviation, and n is the number of data points. The *p*-value is derived from the *t*-score through the *t*-distribution.

²A simple approximate power calculation [171] for fixed 95% confidence level and 90% statistical power is $n = (4rs/\Delta)^2$, where n is the number of users, r is the number of groups, s is the standard deviation, and Δ is the effect to detect.

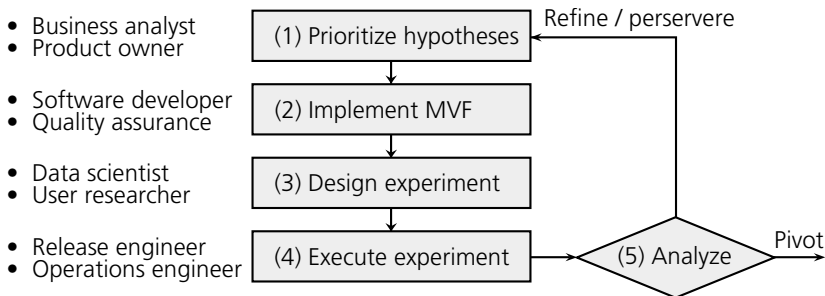


Figure 1: Continuous experimentation process overview in five phases. A hypothesis is prioritized and implemented as a minimum viable product, then an experiment is designed and conducted that evaluates the software change, finally a decision is made to continue or pivot to another feature. This simplified process is based on the RIGHT model. The roles involved in each phase are shown to the left.

The process of continuous experimentation (see Figure 1) has similarities to the tradition from science in software engineering research [310] and elsewhere [112]. However, we base the following process on the RIGHT model by Fagerholm et al. [106]. There are five main phases of the process. (1) In the *ideation* phase hypotheses are elicited and prioritized. (2) *Implementation* of a minimum viable product or feature (MVF) that fulfill the hypothesis follows. (3) Then, a suitable *experiment design* with an OEC is selected. (4) *Execution* involves release engineers deploying the product into production and operations engineers monitoring the experiment in case something goes wrong. Finally, (5) an *analysis* is conducted with either statistical methods by data scientists or by qualitative methods by user researchers. If the results are satisfactory the feature is included in the product and a new hypothesis is selected so the product can be further refined. Otherwise, a decision must be made if to persevere and continue the process or if a pivot should be made to some other feature or hypothesis. Lastly, the results should be generalized into knowledge so the experience gained can be used to inform future hypotheses and development on other features.

Many of the papers included in this study are on improved analysis methods. One such direction that need additional explanation is *segmentation*. It is used in marketing to create differentiated products for different segments of the market. In the context of experiments it is used to calculate metrics for various slices of the data in, e.g., gender, age groups, or country. Experimentation tools usually perform automated segmentation [130] and can, for example, send out alerts if a change affects a particular user group adversely.

2.3 Previous work

Prior to this literature review, two independent mapping studies [9, 247] were conducted by the authors. Although both studies were in the context of continuous experimentation, their objectives differed.

In their mapping study [247], Ros and Runeson provided a short thematic synthesis of the topics in the published research and examined the context of the research in terms of reported organizations and types of experiments that were conducted. They found that there is a diverse spread of organizations of company size, sector, etc. Although, continuous experimentation for software that does not require installation (e.g. websites) was more frequently reported. Concerning the experimentation treatment types, the authors found more reports about visual changes than algorithmic changes. In addition, the least common type of treatment encountered in literature was new features. Finally, it was observed that the standard A/B test was by far the most commonly used experiment design.

The following mapping study [9] by Auer and Felderer investigated the characteristics of the state of research on continuous experimentation. They observed that the intensity of research activities increased from year to year and that there is a high amount of collaboration between industry and academia. In addition, the authors observed that industrial and academic experts contributed equally to the field of continuous experimentation. Concerning the most influential publications (in terms of citations), the authors found that the most common research type among them is experience report. Another observation of the authors was that in total ten different terms were used for the concept of continuous experimentation.

To summarize, the two previous studies discussed continuous experimentation in terms of its applicability in industry sectors, the treatment types and experimentation designs reported, as well as the characteristics of the research in the field. In contrast to these two mapping studies, this study has a far more comprehensive synthesis. Furthermore, the two previous studies improved the rigor and completeness of the search and synthesis procedures.

3 Research Method

Based on these two independently published systematic mapping studies [9, 249] we conducted a joint systematic literature review. Thus, the presented sets of papers from these two studies were used as starting sets. Forward snowballing was applied, by following the assumption from Wohlin [309] that publications acknowledge previous research. Relevant research publications were identified in the resulting sets. Next, the two sets were merged and the resulting set was studied to answer the respective research questions. Therefore, qualitative thematic synthesis [68] and additional narrative synthesis [144] was conducted to answer the research questions based on the found literature.

In the following, the research objective and the forward snowballing procedures are presented. Thereafter, the syntheses used to answer the research questions are described. Finally, the threats to validity are discussed.

3.1 Research objective

The aim of this research is to give an overview of the current state of knowledge about specific aspects of continuous experimentation. The research questions as stated in the introduction are on: (1) core constituents of a CE framework, (2) technical solutions in CE, (3) challenges with CE, and (4) benefits with CE. Based on the prior mapping studies we observed that there were many papers on these topics.

3.2 Forward snowballing

The two existing sets of papers emerging from the previous literature reviews [9, 249], were used as starting sets for forward snowballing. They were selected as starting sets, because both studies were in the field of continuous experimentation and they had similar research directions. Moreover, both studies were conducted within a short time of each other and had similar inclusion criteria. Hence, the authors are confident that the union of both selected paper sets is a good representation of the field of continuous experimentation in this context until 2017.

The forward snowballing was executed independently for each starting set. After having elaborated a protocol to follow, half of the authors worked on Set A (based on [9], with 82 papers) and half of them on the other Set B (based on [249], with 62 papers). In total, the starting sets contained 100 distinctive papers of which 44 papers were shared among both

starting sets. The citations were looked up on Google Scholar³. Since the two previous mapping studies covered publications until 2017, the forward snowballing was conducted by considering papers within the time span 2017–2019. The snowballing was executed until no new publications were found.

In the process of snowballing, we used a joint set of inclusion and exclusion criteria. A paper was included if *any* of the inclusion criteria applies, unless *any* of the exclusion criteria applies. The decision was based primarily on the abstract of papers. If this was insufficient to make a decision, the full paper was examined. In doubt, the selection of a paper was discussed with at least one other author. The criteria were defined as such:

Inclusion criteria

- Any aspect of continuous experimentation (process, infrastructure, technical considerations, etc.)
- Any aspect of controlled experiments (designs, statistics, guidelines, etc.)
- Techniques that complement controlled experiments

Exclusion criteria

- Not written in English
- Not accessible in full-text
- Not peer reviewed or not a full paper
- Not a research paper: track, tutorial, talk, keynote, poster, book
- Duplicated study (the latest version is included)
- Primary focus on business-side of experimentation, advertisement, user interface, recommender system

The quality and validity of the included research publications were ensured through the inclusion and exclusion criteria. For instance, publications that did not go through a scientific peer-reviewing process were not considered according to the exclusion criteria. Moreover, to ensure that only mature work was included both vision papers with no evidence based contribution and short papers with preliminary results were excluded.

³<https://scholar.google.com/>

To summarize, the forward snowballing based on the starting Set A [9] resulted in 100 papers (Set A') and the starting Set B [249] resulted in 88 papers (Set B'). After merging the two paper sets, a total of 128 distinctive papers represent the result of the applied forward snowballing.

3.3 Synthesis

The collection of found papers was studied in more detail to answer each research question. Therefore, a thematic synthesis was conducted for each research question and two of the research questions were additionally answered with qualitative narrative syntheses [144]. The thematic synthesis followed the steps and checklist proposed by Cruzes and Dybå [68] (see Figure 2). Also, the examples of thematic and narrative syntheses given in Cruzes et al. [69] were consulted. Furthermore, all papers were classified in terms of their *research type* according to Wieringa et al. [305] to identify whether they contained solutions (RQ2). Next, the found results were summarized and identified patterns within them were reported. All text segments and codes found in the results of the study are available online (see Section 3.4).

The procedure for the *thematic synthesis* was as follows. As an initial step, all 128 selected papers were read in full and segments of text that relate to a research question were identified. Next, each text segment was labeled with a code. These codes were loosely based on terms that were identified in previous literature studies [9, 249] and evolved during the labeling of the text segments. Thereafter, the codes that had overlapping themes were reduced into themes. In the last step, these themes were arranged according to higher-order themes. The results include 15 higher-order themes in total.

See Figure 2 for an example of the thematic synthesis process, with the data and results for part of the challenges of CE (RQ3). The occurrences of each qualitative data is shown for each step in Figure 2a. A total of 154 segments of text were identified, 84 codes were used, which were reduced to 18 themes, and 6 higher-order themes. The higher-order themes are visible in the structure of the subsections in Section 4. In Figure 2b, examples of each data type is given. The example is based on the *low impact* theme, which was assigned to the higher-order theme *business challenges*.

For the first two research questions, an additional *narrative synthesis* [69] was conducted as a complement to the thematic synthesis. This type of synthesis aggregates qualitative recurring narratives in text. The purpose with conducting the narrative synthesis was to find rationales behind the introduction of infrastructure, processes, or solutions; such that they could be grouped into what problems they were designed to address. This inquiry went beyond the categorization of the thematic synthesis that we applied. In comparison to the thematic synthesis, the analysis procedure was looser but considered the papers in their entirety instead of isolated segments of text.

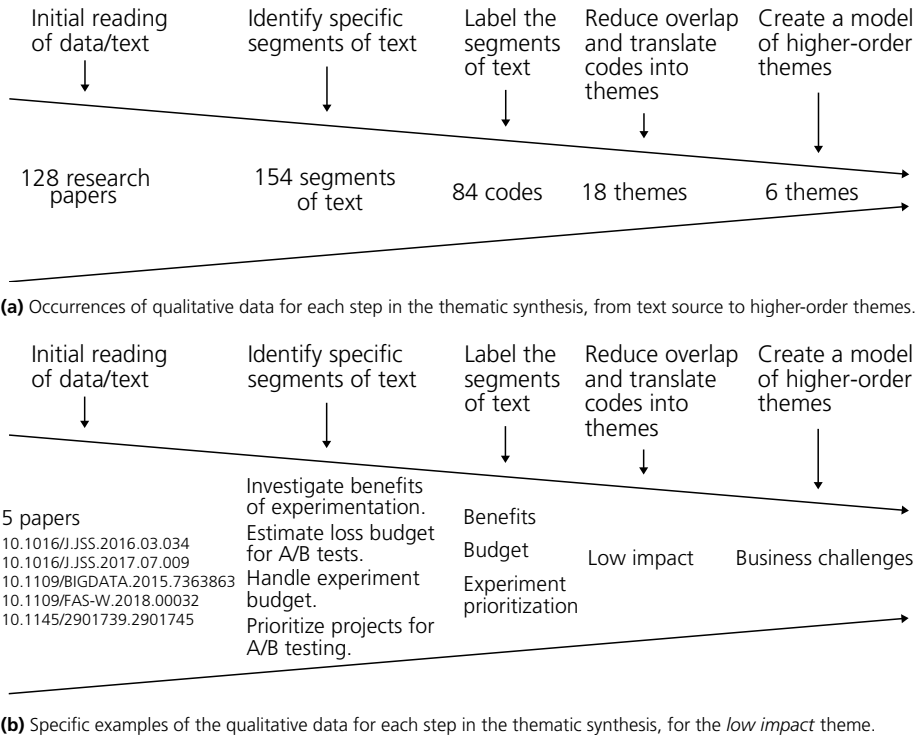


Figure 2: Applied thematic synthesis process (adapted from Cruzes et al. [68, 69]), for RQ3.

3.4 Threats to validity

In every step of this research, possible threats to its validity were considered and minimized when possible. In the following, the potential threats are discussed to provide guidance in the interpretation of this work. This section is structured by the four criteria construct validity, internal validity, external validity and reliability by Easterbrook et al. [91].

3.4.1 Construct validity

This threat is about the validity of identification and selection of publications. A challenging threat to overcome is the completeness of the literature search without a biased view of the subject. To mitigate this threat, all papers from the start sets were used without any further exclusion. The larger start set (in comparison to applying the exclusion criteria from the start) was expected to lead to a broader coverage of the literature during the forward snowballing. Furthermore, the process of forward snowballing was adapted in a way such that the candidate selection was tolerant about which papers to include, thereby increasing

the coverage of the literature search. However, publications may have been falsely excluded because of misjudgment. Nevertheless, we conducted two parallel forward snowballing searches by different authors based on slightly different starting sets, which should mitigate this threat.

3.4.2 Internal validity

Threats that are caused by faulty conclusions could happen because of authors bias at the selection, synthesis of publications, and interpretation of the findings. To mitigate this threat, a second author was consulted in case of any doubt. Nevertheless, activities like paper inclusion or exclusion and thematic synthesis inevitably suffer from subjective decisions.

3.4.3 External validity

Threats to external validity covers to which extent the generalization of the results is justified to other companies, fields, etc. As the aim of this study is to give an overview of continuous experimentation and to explore the future work items in continuous experimentation, the results should not be generalized beyond continuous experimentation. Therefore, this threat is negligible.

3.4.4 Reliability

This threat focuses on the reproducibility of the study and the results. To mitigate this threat every step and decision of the study were recorded carefully and the most important decisions are reported. The results of the study are available online [12]. This enables other researchers to validate the decision made on the data. Furthermore, it increases transparency and allows theoretical repeatability of the study.

4 Results

In this section the results of the literature review are presented according to the research questions.

4.1 What are the core constituents of a CE framework (RQ1)?

To conduct continuous experimentation, an organization has to have some constituents of a framework for experimentation. There is some process involved (implicit or explicit) and some infrastructure is required, which includes a toolchain as well as organizational processes. In the following, both aspects of an experiment are discussed in detail: the process and its supporting infrastructure.

4.1.1 Experiment process

The experiment process can be described in a model that gives a holistic view of the phases and environment around experimentation. Most studies on experiment processes present models based on qualitative interview data. Two models describe the overall process of experimentation. First, the reference model RIGHT (Rapid Iterative value creation Gained through High-frequency Testing) by Fagerholm et al. [106] contains both an infrastructure architecture and a process model for continuous experimentation. The process model builds on the Build, Measure, Learn [240] cycle of Lean Startup. The process in Figure 1 is a simplified view of RIGHT. Second, the HYPEX (Hypothesis Experiment Data-Driven Development) model is another earlier process model by Holmström Olsson and Bosch [220]. In comparison to the RIGHT model, it is less complete in scope, however it does go into further details in hypothesis prioritization using a gap analysis.

Kevic et al. [164] present concrete numbers on the experiment process used at Microsoft Bing through a source code analysis. They have three main findings: (1) code associated with an experiment is larger in terms of files in a changeset, number of lines, and number of contributors; (2) experiments are conducted in a sequence of experiments lasting on average 42 days, where each experiment is on average conducted for one week; and (3) only a third of such sequences are eventually shipped to users.

In addition to the general models described above, several models deal with a specific part of the experiment cycle. The CTP (Customer Touchpoint) model by Sauvola et al. [259] focuses on user collaboration and describes the various ways that user feedback can be involved in the experimentation stages. Amatrian [5] and Gomez-Uribe and Hunt [125] describe their process for experimentation on their recommendation system at Netflix, in particular, how they use offline simulation studies with online controlled experiments. In the ExG Model (Experimentation Growth), by Fabijan et al. [101, 103], organizations can quantitatively gauge their experimentation on technical, organizational, and business aspects. In another model by Fabijan et al. [102] they describe the process of analyzing the

results of experiments and present a tool that can make the process more effective, by e.g., segmenting the participants automatically and highlighting the presence of outliers. Finally, Mattos et al. [198] present a model that discusses details on activities and metrics on experiments.

4.1.2 Infrastructure

Depending on what type of experimentation is conducted, different infrastructure is required. For controlled experimentation, in particular, technical infrastructure in the form of an *experimentation platform* is critical to increase the scale of experimentation. At the bare minimum it needs to divide users into experiment groups and report statistics. Gupta et al. [130] at Microsoft have detailed the additional functionality of their experimentation platform. Also, Schermann et al. [265] have described attributes of system and software architecture suitable for experimentation, namely, that micro-service-based architectures seem to be favored. Some experimentation platforms are specialized to specific needs: automation [196], or describing deployment through formal models [263], or how experimentation can be supported by non-software engineers [177, 111].

There are also non-technical infrastructure requirements, regardless of the type of experimentation in use. The required roles are [113]: data scientists, release engineers, user researchers, and the standard software engineering roles. Also, an organizational culture [176, 315] that is open towards experimentation is needed. For example, Kohavi et al. [176] explain that managers can hinder experimentation if they overrule results with their opinions. They call the phenomenon the highest paid persons opinion (HiPPO).

While experimentation is typically associated with large companies, like Microsoft or Facebook, there are three interview studies that discuss experimentation at startups specifically [27, 106, 131]. As argued by Gutbrod et al. [131], startup companies often guess or estimate roughly about the problems and customers they are addressing. Thus, there is a need for startup companies to be more involved with experimentation, even though they have less infrastructure in place.

Finally, we would like to call attention to some of the few case studies and experience reports on experimentation on “ordinary” software companies, which are neither multi-national corporations nor startups; in e-commerce [247], customer relations [242], and game development [321]. None of these papers are focused on infrastructure, but do mention that infrastructure needs to be implemented. Though, Risannen et al. [242] mentions additional challenges when infrastructure must be implemented on top of a mature software product. In summary, this indicates that infrastructure requirements are modest, unless scaling up to multi-national corporation levels with millions of users.

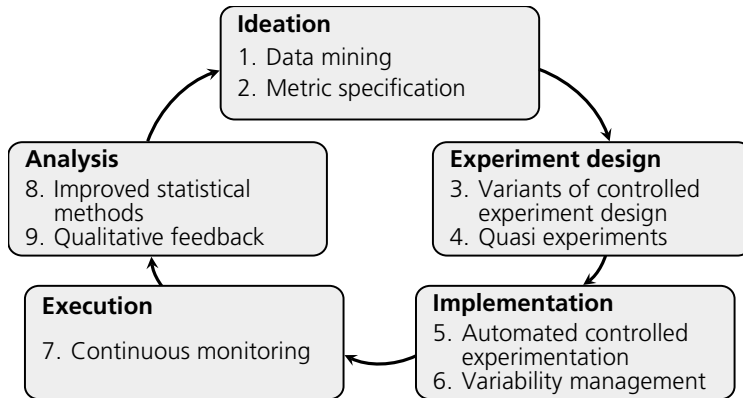


Figure 3: Solutions applied within continuous experimentation arranged by main phase of experimentation from Section 2.2.

4.2 What technical solutions are applied in what phase within CE (RQ2)

The study of the selected publications revealed many different types of solutions that were summarized by common themes. Figure 3 gives an overview of the identified solutions organized in the phases of experimentation from Figure 1.

4.2.1 Data mining

Data from previous experiments can be used to make predictions or mine insights to either improve the *reliability* of the experiment or for *ideation*. There were three specific solutions for data mining in continuous experimentation. (1) Statistical tests can be improved by calculating variance of metrics through bigger data sets than just one experiment; this is in place at Netflix [312], Microsoft [74, 77], Google [140], and Oath [7]. (2) Mining for invalid tests through automatic diagnosis rules is in place at LinkedIn [56] and Sevenval Technologies [218]. Finally, (3) data mining is used to extract insights from segments of the users, by detecting if a treatment is more suitable for those specific circumstances [89]; this technique is applied at Snap [313] and Microsoft [102].

4.2.2 Metric specification

Defining metrics for software is difficult. This has been studied primarily at Microsoft [76, 81, 193] and Yandex[45, 88, 167]. Some general *guidelines* for defining metrics follow. At Microsoft, they have hundreds of metrics for each experiment (in addition to a few OEC). Machmouchi and Buscher [193] from Microsoft describe how their metrics are interpreted in a hierarchy in their tool (similar to Fabijan et al. [102] also at Microsoft). At the top of the hierarchy are statistically robust metrics (meaning they tend not to give false positives)

and at the bottom are feature specific metrics that are allowed to be more sensitive. At Yandex, they always pair OEC metrics with a statistical significance test to create an overall acceptance criteria (OAC) instead [87]; thereby reducing the risk of incorrectly pairing significance tests to metrics.

There are techniques for *evaluating metrics*, in how well they behave in experiments. Dmitriev et al. [81] give an experience report on how metrics are continuously evaluated at Microsoft system in practice. Deng et al. [76] define metrics for evaluating metrics: directionality and sensitivity. They measure, respectively, whether a change in the metric aligns with good user experience and how often it detects a change in user experience.

Usability metrics are hard to define since they are not directly measurable without specialized equipment, such as eye-tracking hardware or focus groups. The measurements that are available, such as clicks or time spent on the site, do not directly inform on whether a change is an improvement or degradation in user experience. In addition, good user experience does not necessarily correlate positively with business value, e.g., clickbait titles for news articles are bad user experience but generate short term revenue. Researchers from Yandex [45, 85, 86, 88, 167, 234] are active in this area, with the following methods focused on usability metrics: detecting whether a change in a metric is a positive or negative user experience [85]; learning sensitive combinations of metrics [167]; quantifying and detecting trends in user learning [88]; predicting future behavior to improve sensitivity [86]; applying machine learning for variance reduction [234]; and finally correcting misspecified usability metrics [45]. Machmouchi et al. [194], at Microsoft, designed a rule-based classifier where each user action is either a frustration or benefit signal; the tool then aggregates all such user actions taken during a session into a single robust utility metric.

4.2.3 Variants of controlled experiments design

Most documented experiments conducted in industry are univariate A/B/n-tests [249], where one or more treatments are tested against a control. Extensions to classical designs include a two-staged approach to A/B/n tests [78] and a design to estimate causal effects between variables in a multivariate test (MVT) [231]. MVTs are cautioned against [175] because of their added complexity. In contrast, other researchers take an optimization approach using lots (see Section 4.2.5) of variables with multi-armed bandits [62, 138, 199, 247] or search-based methods [208, 250, 285]. Also, mixed methods research is used to combine quantitative and qualitative data (see the solution theme on *Qualitative feedback*).

4.2.4 Quasi-experiments

A quasi-experiment (or natural experiment) is an experiment that is done sequentially instead of in parallel; this definition is the same as in empirical research in general [310]. The reason for doing it is that it has a lower technical complexity. In fact, any software deployment can have its impact measured by observing the effect before and after deployment. The drawback of this is that analyzing the results can be difficult due to the high risk of having external changes affect the result. That is, if anything extraordinary happens roughly at the same time as the release it might not be possible to properly isolate the results. Since the world of software is in constant change the use of quasi-experiments is challenging. The research directions on quasi-experiments involve how to eliminate external sources of noise to get more reliable results. This is studied at Amazon [137] and LinkedIn [314], particularly for environments where control in continuous deployment is hard (such as mobile app development).

4.2.5 Automated controlled experimentation with optimization algorithms

With an optimization approach, the allocation of users to the treatment groups is dynamically varied to optimize an OEC, such that treatments that perform well continuously receive more and more traffic over time. With sufficient automation, these techniques can be applied to lots of treatment variables simultaneously. This is not a replacement for classical designs; in an interview study by Ros and Bjarnason [247], they explain that such techniques are often validated themselves using A/B tests. In addition, based on the studies included here, only certain parameters are eligible, such as the design and layout of components in a GUI, or parameters to machine learning algorithms or recommender systems. Some of these optimizations are black-box methods, where multiple variables are changed simultaneously and with little opportunity to make statistical inferences from the experiments.

Tamburelli and Margara [285] proposed search-based methods (i.e. genetic algorithms) for optimization of software, and Iitsuka and Matsuo [148] demonstrated a local search method with a proof of concept on web sites. Miikkulainen [208], at Sentient Technologies, have a commercial genetic algorithm profiled for optimizing e-commerce web sites. Bandit optimization algorithms are also used in industry at Amazon [138] and AB Tasty [62], it is a more rigorous formalism that requires the specification of a statistical model on how the OEC behaves. Ros et al. [250] suggested a unified approach of genetic algorithms and bandit optimization. Similar algorithms exist to handle continuous variables, as is needed for hardware parameters [121, 199] and for optimizing machine learning and compiler parameters [184].

Two studies apply optimization [165, 262] to scheduling multiple standard A/B tests to users, where only a single treatment is administered to each user. The idea is to optimize an OEC without sacrificing statistical inference.

4.2.6 Variability management

Experimentation incurs increased variability—by design—in a software system. This topic deals with solutions in the form of tools and techniques to manage said variability. In terms of an experiment platform, this can be part of the *experiment execution service* or the *experimentation portal* [130].

There have been attempts at imposing systematic constraints and structure in the configuration of how the variables under experimentation interact with formal methods. Cámara and Kobsa [48] suggest using a feature model of the software parameters in all experiments. This work has not advanced beyond a proof-of-concept stage.

Neither in our study, nor in the survey by Schermann et al. [265], is there any evidence of formal methods in a dynamic and constantly changing experimentation environment. The focus of the tools in actual use are rather on flexibility and robustness [15, 286]. Rahman et al. [236] studied how feature toggles are used in industry. Feature toggles are ways of enabling and disabling features after deployment, as such they can be used to implement A/B testing. They were found to be efficient and easy to manage but adds technical debt.

A middle ground between formal methods and total flexibility has evolved in the tools employed in practice. Google has proprietary tools in place to manage overlapping experiments in large scale [286]. In their tools, each experiment can claim resources used during experimentation and a scheduler ensures that experiments can run in parallel without interference. Facebook has published an open-source framework (PlanOut) specialized for configuring and managing experiments [15], it features a namespace management system for experiments running iteratively and in parallel. SAP has a domain-specific language [302] for configuring experiments that aims at increasing automation. Finally, Microsoft has the ExP platform, but none of the selected papers focus solely on the variability management aspect of it.

4.2.7 Improved statistical methods

The challenges with experimentation motivate improved statistical techniques specialized for A/B testing. There are many techniques for fixing specific biases, sources of noise, etc. We found eight specific solutions for this theme: (1) a specialized test for count data at SweetIM [30]; (2) fixing errors with dependent data at Facebook [16]; (3) improvements

from the capabilities of A/A testing on diagnosis (which tests control vs control expecting no effect) at Yahoo [323] and Oath [57]; (4) better calculation of overall effect for features with low coverage at Microsoft [75]; (5) fixing errors from personalization interference at Yahoo [71]; (6) fixing tests under telemetry loss at Microsoft [129]; (7) correcting for selection bias at Airbnb [183]; and (8) algorithms for improved gradual ramp-up at Google [204] and LinkedIn [316].

4.2.8 Continuous monitoring

Aborting controlled experiments pre-maturely in case of outstanding or poor results is a hotly debated topic on the internet and in academia, under the name of continuous monitoring, early stopping, or continuous testing. The reason for wanting to stop early is to reduce opportunity costs and to increase development speed. It is studied by Microsoft [79], Yandex [166], Optimizely [153], Walmart [2], and Etsy [155]. This concept is similar to the continuous monitoring used by researchers in the DevOps community and continuous software engineering [113] where it refers to the practice of monitoring a software system and sending alerts in case of faults. The issue with continuous monitoring of experiments is the increased chance of getting wrong results if carried out incorrectly. Traditionally, the sample size of an experiment is defined beforehand through a power calculation. If the experiment is continuously monitored with no adjustment, then the results will be skewed with inflated false negative and positive error rates.

4.2.9 Qualitative feedback

While the search strategy in this work was focused on controlled experiments, research on qualitative feedback was also included from experience reports on using many different types of feedback collecting methods, for example at Intuit [31, 32] and Facebook [107]. The qualitative methods are used as complements to quantitative methods, either as a way to better explain results or as a way to obtain feedback earlier in the process, before a full implementation is built. That is, qualitative feedback can be collected on early user experience sketches or mock-ups. Another use of qualitative methods is to elicit hypotheses that can be used as a starting point for an experiment. Examples of methods include focus groups, interviews, and user observations.

In addition, at Unister [277] the authors explain how they collect qualitative user feedback in parallel with A/B tests, such that the feedback is split by experiment group. According to the authors, this seems to be a way to get the best of both quantitative and qualitative worlds. It does require implementing a user interface for collecting the feedback in a non-intrusive way in the product. Also, the qualitative feedback will not be of as high quality as when it is done in person with, e.g., user observation or focus groups.

4.3 What are the challenges with continuous experimentation (RQ3)?

Continuous experimentation encompasses a lot of the software engineering process, it requires both infrastructure support and a rigorous experimentation process that connects the software product with business value. As such, many things can go wrong and the challenges presented here is an attempt at describing such instances. Most of the research on challenges is evaluation research, with interviews or experience reports. The analysis of the papers revealed five categories of challenges (see Table 1) that are discussed below in more detail. Many of the challenges are severe, in that they present a hurdle that must be overcome to conduct continuous experimentation. A failure in any of the respective category of challenges will make an experiment: (1) not considered by unresponsive management, (2) without a business case, (3) unfeasible due to technical reasons, (4) untrustworthy due to faulty use of statistics, or (5) ethically questionable. In addition, we identified additional challenges in four domain specific clusters.

4.3.1 Cultural, organizational, and managerial challenges

The challenges to organizations and management are broad in scope, with four specific challenges: (1) *knowledge building* of experimentation skills among employees across the whole organization [174, 320]; (2) leadership practicing *micromanagement*, who disregard experiment results in favor of their own opinion [171, 176]; (3) difficulty in changing the organizational culture to *adopt experimentation* among engineers [188]; and finally (4) *communicating results* and coordinating experiments in business to business, where there are stakeholders involved across multiple organizations [242, 320].

The second challenge, *micromanagement*, is a fundamental challenge that has to be faced by organizations adopting continuous experimentation. That is, the shift from the highest-paid person's opinion (HiPPO) [171, 176] to data-driven decision making. If managers are used to making decisions about the product then they might not take account of experimental results that might run counter to their intuition. Thus, decision-makers must be open to have their opinions changed by data, else the whole endeavor with experimentation is useless.

Table 1: Summary of challenges with CE per category with description and key references that focus on them.

Challenge	Description	References
<u>1. Cultural, organizational, and managerial challenges</u>		
Knowledge building	There are many roles and skills required, so staff need continuous training.	[174, 242, 320]
Micromanagement	Experimentation requires management to focus on the process (c.f. HiPPO in Section 4.3.1).	[176]
Lack of adaption	Engineers need to be onboarded on the process as well as managers.	[188]
Lack of communication	Departments and teams should share their results to aid each other.	[242, 320]
<u>2. Business challenges</u>		
Low impact	Experimentation might focus efforts on incremental development with insufficient impact.	[113, 224]
Relevant metrics	The business model of a company might not facilitate easy measurement.	[83, 103, 188]
Data leakage	Companies expose internal details about their product development with experimentation.	[66]
<u>3. Technical challenges</u>		
Continuous delivery	The CI/CD pipeline should be efficient to obtain feedback fast.	[103, 188, 265]
Continuous deployment	Obstacles exists to putting deliveries in production, e.g. on-premise installations in B2B.	[242]
Experimental control	Dividing users into experimental groups have many subtle failure possibilities.	[67, 82, 172]
<u>4. Statistical challenges</u>		
Exogenous effects	Changes in environment can impact experiment results, e.g. trend effects in fashion.	[82, 173]
Endogenous effects	Experimentation itself causes effects, such as carry-over or novelty effects.	[82, 190]
<u>5. Ethical challenges</u>		
Data privacy	GDPR gives users extensive rights to their data which companies must comply with.	[319]
Dark patterns	A narrow focus on numbers only can lead to misleading user interfaces.	[150]
<u>6. Domain specific challenges</u>		
Mobile	The app marketplaces impose constraints on deployment and variability.	[185, 314, 321]
Cyber-physical systems	Making continuous deployments can be infeasible for cyber-physical systems.	[34, 123, 197]
Social media	Users of social media influence each other which impacts the validity of experiments.	[13, 14, 61]
E-commerce	Experimentation needs to be able to differentiate effects from products and software changes.	[126, 300]

4.3.2 Business challenges

The premise behind continuous experimentation is to increase the business value of software development efforts, which is challenging for three reasons. First, defining relevant metrics that measure business value is the most frequent challenge in realizing business value [82, 83, 103, 188, 320]. In some instances the metric is only indirectly connected to business, for example in a business-to-business (B2B) company with a revenue model that is not affected by the software product, then improving product quality and user experience will not have a direct business impact.

Second, the impact of experiments might not be sufficient in terms of actual effect [175, 224]. Fitzgerald and Stol [113] argue that continuous experimentation and innovation can lead to incremental improvements only, at the expense of more innovative changes that could have had a bigger impact.

Finally, *data leakage* was highlighted by Conti et al. [66]; they crawled web sites repeatedly and tried to automatically detect a difference in server responses. Thereby they showed how easily such data leakage can facilitate industrial espionage on what competitors are developing.

4.3.3 Technical challenges

Efficient continuous deployment facilitates efficient experimentation. Faster deployment speed shortens the delay between a hypothesis and the result of an experiment. The ability to have an efficient *continuous delivery* cycle is cited as a challenge both for large [172] and small companies [103, 188, 265]. In addition, *continuous deployment* is further complicated in companies involved in business to business (B2B) [242], where deployment has multiple stakeholders involved over multiple organizations.

In a laboratory experiment setting, it is possible to control variables such as ensuring homogeneous computer equipment for all groups and ensuring that all groups have equal distribution in terms of gender, age, education, etc. For online experiments, such *experimental control* is much harder due to subtle technical reasons. Examples of that is: users assigned incorrectly to groups due to various bugs [172]; users changing groups because they delete their browsing history or multiple persons share the same computer [64, 80, 82, 172]; and web crawler robots from search engines cause abnormal traffic affecting the results [67, 170].

4.3.4 Statistical challenges

Classical experimental design as advocated by the early work on continuous experimentation and A/B-testing [171] does not account for time series. Not only can it be hard to detect the presence of effects related to trends, but they can also have an effect on the results. Some of these trend effects occur due to outside influence, so-called *exogenous effects*, for example, due to seasonality caused by fashion or other events which can affect traffic [82, 173]. With domain knowledge, these effects can be accounted for. For example in e-commerce, experiment results obtained during the weeks before Christmas might not transfer to the following weeks.

Other statistical challenges are caused by the experimentation itself, called *endogenous effects*, such as the carryover effect [173, 190] where the result of an experiment can affect the result of a following experiment. There are also endogenous effects caused intentionally, through what is known as ramp up, where the traffic to the test group is initially low (such as 5%/95%) and incrementally increased to the full 50%/50% split. This is done to minimize the opportunity cost of a faulty experiment design. It can be difficult to analyze the results of such experiments in a dynamic environment such as software engineering [67, 170]. Furthermore, learning and novelty effects where the users change their impression of the feature after using it for a while are also challenging [82, 190].

Endogenous effects will be hard to foresee until experimentation is implemented in a company. As such, handling statistical challenges is an ongoing process that will require more and more attention as experimentation is scaled up.

4.3.5 Ethical challenges

Whenever user data is involved there is a potential for ethical dilemmas. When Yaman et al. [319] surveyed software engineering practitioners, the only question they agreed on was that users should be notified if personal information is collected. Since GDPR went into effect in 2018 the right to *data privacy* is now a requirement. Generally, users agree to partake in experiments through accepting the terms of service, as stored in cookies. Jian et al. [150] investigate how A/B testing tools are used in illegal discrimination for certain demographics, e.g., by adjusting prices or filtering job ads. These are examples of what is known as *dark patterns* in the user experience (UX) research community [127]. The study by Jian et al. was limited to sites using front end Optimizely (a commercial experimentation platform) metadata.

4.3.6 Domain specific challenges

Some software sectors have domain-specific challenges or techniques required for experimentation, of which in the analysis of the papers four prominent domains were found: (1) *mobile apps*, (2) *cyber-physical systems*, (3) *social media*, and (4) *e-commerce*. Whether or not all of these concerns are domain-specific or not is debatable. However, these studies were all clear on what domain their challenges occurred in and challenges were presented as domain specific.

There is a bottleneck in continuous deployment to the proprietary application servers of Android Play or Apple's App Store, which imposes a bottleneck on experimentation for *mobile apps*. Lettner et al. [185] and Adinata and Liem [3] have developed libraries that load new user interfaces at run time, which would otherwise (at the time of writing in year 2013 and 2014, respectively) require a new deployment on Android Play. Xu et al. [314] at LinkedIn instead advocate the use of quasi-experimental designs. Finally, Yaman et al. [321] have done an interview study on continuous experimentation, where they emphasize user feedback in the earlier stages of development (that do not require deployment).

Cyber-physical systems, embedded systems, and smart systems face similar challenges as mobile apps, namely continuous deployment. None of the studied publications of this study claims widespread adoption of experimentation at an organizational level. This suggests that research of experimentation for embedded software is in an early stage. Mattos et al. [197] and Bosch and Holmström Olsson [34] outline challenges and research opportunities in this domain, among them are: continuous deployment, metric definition, and privacy concerns. Bosch and Eklund [33, 94] describe required architecture for experimentation in this domain with a proof-of-concept on vehicle entertainment systems. Gaiimo et al. [123, 122] cite safety concerns and resource constraints for the lack of continuous experimentation.

The cyber-physical systems domain also includes experimentation where the source of noise is not human users, but rather hardware. The research on self-adaptive systems overlap with continuous experimentation: Gerostathopoulos et al. [118] have described an architecture for how self-adaptive systems can perform experimentation, with optimization algorithms [120] that can handle non-linear interactions between hardware parameters [119]. In addition, two pieces of work [44, 149] on distributed systems focus on experimentation, with a survey and a tool on how distributed computing can support experimentation for e.g. cloud providers.

Backstrom et al. [14] from Facebook describe that users of *social media* influence each other across experiment groups (thus violating the independence assumption of statistical tests); they call it the network effect. It is also present at Yahoo [161] and LinkedIn [128, 260, 315]. The research on the network effect includes: ways of detecting it [260], estimating its effect on cliques in the graph [13, 61], and reducing the interference caused from it [93].

The final domain considerations come from *e-commerce*. At Walmart, Goswami et al. [126] describe the challenges caused by seasonality effects during holidays and how they strive to minimize the opportunity cost caused by experimentation. At Ebay, according to Wang et al. [300], the challenges are caused by the large number of auctions that they need to group with machine learning techniques for the purpose of experimental control.

4.4 What are the benefits with continuous experimentation (RQ4)?

The implicit benefits of CE is that the quality of the product is increased in relation to the metric used in experiments. Many authors mention this implicit benefit of CE only in passing as motivation for the research, only one paper explicitly focus on additional benefits.

4.4.1 Implicit benefits

Bosch [31] mentions the reduced cost of collecting passive customer feedback with continuous experimentation in comparison with active techniques like surveys. Also, Bosch claims that customers have come to expect software services to continuously improve themselves and that experimentation can provide the means to do that in a process that can be visible to users. Kohavi et al. [176] claim that edge cases that are only relevant for a small subset of users can take a disproportionate amount of the development time. Experimentation is argued for as a way to focus development, by first ensuring that a feature solves a real need with a small experiment and then optimizing the respective feature for the edge cases with iterative improvement experiments. In this way, unnecessary development on edge cases can be avoided if a feature is discarded early on.

4.4.2 Explicit benefits

Fabijan et al. [100] focus solely on additional benefits of CE, differentiated between three levels as follows. (1) In the *portfolio level*, the impact of changes on the customer as well as business value can be measured which is of great benefit to company-wide product portfolio development. (2) In the *product level*, the product receives incrementally improvement quality and reduced complexity by removing unnecessary features. Finally, (3) in the *team*

level of benefits, the findings of experiments support the related teams to prioritize their development activities given the lessons learned from the conducted experiments. Another benefit for teams with continuous experimentation is that team goals can be expressed in terms of metric changes and their progress is measurable.

5 Discussion

This study builds on two prior independent mapping studies to provide an overview of the conducted research. This review has been conducted to answer four research questions that can guide practitioners. In the following, the results of the study are discussed for each research question, in the form of recommendations to practitioners and implications for researchers.

5.1 Required frameworks (RQ1)

The first research question (RQ1) about the core constituents of a framework for continuous experimentation revealed two integral parts of experimentation, the *experimentation process* and the technical as well as organizational *infrastructure*.

5.1.1 Process for continuous experimentation

In the literature, several experimentation process models were found on the phases of conducting online controlled experimentation. They describe the overall process [106], represent the established experiment process of organizations [164], or cover specific parts of the experiment cycle [102]. Given that all models describe a process with the same overall objective of experimentation, it can become difficult to decide between them. The reference models may be used as a basis for future standardization of the field [5, 106, 125, 220]. Future research is needed to give guidance in the selection between reference models and variants [102, 198, 259].

Many of the experience reports [175, 170] warn about making experiments with too broad scope, instead they recommend that all experiments should be done on a minimum viable product or feature [106]. However, the warnings all come from real lessons learned caused by having done such expensive experiments. We believe that the current process models do not put sufficient emphasis on conducting small experiments. For example, they could make a distinction between prototype experiments and controlled experiments on a completed project. That way, if the prototype reveals flaws in the design it avoids a full implementation.

As such, our recommendation to practitioners in regards to process is to follow one of the reference experimentation processes [106, 220] and in addition add the following two steps to minimize the cost of experiments. First, to spend more time before experimentation to ensure that experiments are really on a minimum viable feature by being diligent about what requirements are strictly needed at the time. Second, that experiments should be pre-validated with prototypes, data analysis, etc.

5.1.2 Infrastructure for continuous experimentation

The research on the infrastructure required to enable continuous experimentation was primarily focused on large scale applications within mature organizations (e.g. Microsoft [130]). The large number of industrial authors indicates a high practitioner interest in the topic. However, it should not restrict the community's focus on large scale applications only. The application of continuous experimentation within *smaller organizations* has many open research questions. These organizations have additional challenges to experiment, because of their likely smaller existing user base and infrastructure. Also, the development of sophisticated experimentation platforms may not be feasible in the extent to which it is for large organizations. Thus, lightweight approaches to experimentation that do not require large up-front investments could make experimentation more accessible to smaller organizations.

Technical infrastructure has not been reported as being a significant hurdle for any of the organizations in which continuous experimentation was introduced in this study. The technical challenges seem to appear later on when the continuous experimentation process has matured and the scale of experimentation needs to ramp up. Rather, the *organizational infrastructure* seems to be what might cause an inability to conduct experimentation. The challenges presented in Section 4.3 support this claim too, so the more severe infrastructural requirements appear to be organizational [113] and culture oriented [176, 315], at least to get started with experimentation. The reason for this is that experimentation often involves decision making that traditionally fall outside the software development department. For example, deciding on what metric software should be optimized for might even need to involve the company board of directors. Following that, the recommendation to practitioners is to not treat continuous experimentation as a project that can be solved with only software development. The whole organization needs to be involved, e.g., to find metrics and to ensure that the user data to measure this can be acquired. Otherwise, if the software development organization conducts experimentation in isolation, the soft aspects of infrastructure might be lacking or the software might be optimized with the wrong goal in mind.

5.2 Solutions applied (RQ2)

Concerning the solutions that are applied within continuous experimentation (RQ2), themes were proposed for each of the solutions in literature. The literature analysis revealed solutions about qualitative feedback, variants of controlled experiments design, quasi-experiments, automated controlled experimentation with optimization algorithms, statistical methods, continuous monitoring, data mining, variability management, and metric specification. One observation made was that the validation of solutions proposals could be further improved by providing the used data sets and a context description or the necessary steps that allow to reproduce the presented results. Also, many interesting solutions would benefit from further applications that demonstrate their applicability in practice. Another observation was that many solutions are driven by practical problems of the author's associated organization (e.g., evaluation of mobile apps [137]). This has the advantage that the problems are of relevance for practice and the provided solutions are assumed to be applicable in similar contexts. Publications of this kind are guidelines for practitioners and valuable research contributions.

There are a lot of solutions for practitioners to choose from, most of them solve a very specific problem that has been observed at a company. In Figure 3, the solutions are arranged by phase of the experimentation process. What follows is additional help to practitioners to know what solution to apply for a given problem encountered, which is in the design science tradition known as technological rules [98]:

- to achieve *additional insights* in concluded experiments apply (1) *data mining* that automatically segments results for users' context;
- to achieve *more relevant results* in difficult to measure software systems apply (2) *metric specification techniques*.
- to achieve *richer experiment feedback* in continuous experimentation apply (3) *variants of controlled experiments design* or (9) *qualitative feedback*.
- to achieve *quantitative results* in environments where parallel deployment is challenging apply (4) *quasi-experiments*;
- to achieve *optimized user interfaces* in software systems that can be evaluated on a single metric apply (5) *automated controlled experimentation with optimization algorithms*;
- to achieve *higher throughput of experiments* in experimentation platforms apply (6) *variability management techniques* to specify overlapping experiments;
- to achieve *trustworthy results* in online controlled experiments apply (7) *improved statistical methods* or (1) *data mining* to calibrate the statistical tests;

- to achieve *faster results* in online controlled experiments apply (8) *continuous monitoring* to help decide when experiments can be stopped early.

5.3 Challenges (RQ3)

Many authors of the studied literature mentioned challenges with continuous experimentation in their papers. The thematic analysis of the challenges identified six fundamental challenge themes. Here they are presented along with the recommendations to mitigate the risks.

The *cultural, organizational and managerial* challenges seem to indicate that the multi-disciplinary characteristic of continuous experimentation introduces new requirements to the team. It requires amongst others the collaboration of cross-functional stakeholders (i.e. business, design, and engineering). This can represent a fundamental cultural change within an organization. Hence, the adaption of continuous experimentation involves technical as well as cultural changes. Challenges like the lack of adaption support this interpretation. Mitigating these challenges involves taking a whole organizational approach to continuous experimentation so that both engineers and managers are in agreement about conducting experimentation.

Another theme among challenges is *business*. The challenges assigned to this theme highlight that continuous experimentation has challenges in its economic application with respect to the financial return on investment. The focus of experimentation needs to be managed appropriately in order to prevent investing in incremental development with insufficient impact. Also, that changes cannot be measured with a relevant metric is another business challenge. One possible approach for further research on these challenges could be the transfer from solutions in other disciplines to continuous experimentation. An example therefore is the overall evaluation criteria [252] that was adapted to continuous experimentation by Kohavi et al. [171]. As with the previous challenge theme, this theme of challenges does not have an easy fix. It might be the case that experimentation is simply not applicable for all software companies but further research is needed to determine this.

Concerning the *technical* challenges, the literature review showed that there are challenges related to continuous deployment/delivery and experiment control. The delivery of changes to production is challenging especially for environments that are used to none or infrequent updates, like embedded devices. For such edge cases, new deployment strategies have to be found that are suitable for continuous experimentation. Although solutions from continuous deployment seem to be fitting, they need to be extended with mechanisms to control

the experiment at run-time (e.g. to stop an experiment). This can be challenging in environments for which frequent updates are difficult. There is proof-of-concept research [94] to handle these challenges so they do not seem to be impossible blockers to get started on experimentation.

The *statistical* challenges mentioned in the studied literature indicate that there is a need for solutions to cope with the various ways that the statistical assumptions done in a controlled experiment are broken by changes in the real world. There are both changes in the environment (exogenous) and changes caused by experimentation (endogenous). Changes in the environment (e.g. the effect of an advertisement campaign run by the marketing department) can alter the initial situation of an experiment and thus may lead to wrong conclusions about the results. Therefore, the knowledge about an experiment's environment and possible influences needs to be systematically advanced and the experiments themselves should be designed to be more robust. Mitigating these challenges involves identifying and applying the correct solution for the specific problem. There is further research opportunity to document and synthesize such problem-solution pairs.

Ethical aspects are not investigated by many studies. The experience reports and lessons learned publications do not, for example, mention user consent or user's awareness of participation. Furthermore, ethical considerations about which experiments should be conducted or not were seldom discussed in the papers. There were still two challenges identified in this study, involving data privacy and dark patterns. However, examples like the Facebook emotional manipulation study, which changed the user's news feed to determine whether it affects the subsequent posts of a user, show the need for ethical considerations in experimentation [114]. Although this was an experiment in the context of an academic study in psychology, the case nevertheless shows that there are open challenges on the topic of ethics and continuous experimentation. There is not enough research conducted for a concrete recommendation other than raising awareness of the existence of ethical dilemmas involving experimentation.

Continuous experimentation is applied in various domains that require *domain specific* solutions. The challenges on continuous experimentation range from infrastructure challenges, over measurement challenges, to social challenges. Examples are the challenge to deploy changes in cyber-physical systems (technical challenge), to differentiate the effects of one change from another (statistical challenge), and the influence of users on each other across treatment groups (statistical challenge). Each challenge is probably only relevant for certain domains, however the developed solutions may be adaptable to other domains. Thus, the research on domain-specific challenges could take optimized solutions for specific domains to other domains.

5.4 Benefits (RQ4)

In many publications about continuous experimentation the benefits of experimentation are mentioned as motivation only; i.e., it increases the quality of the product based on the chosen metrics. The two publications on explicit benefits [31, 100] mention improvements not only on the product in business-related metrics and usability but also on the product portfolio offering and generic benefits for the whole organization (better collaboration, prioritization, etc.). More studies are needed to determine, e.g., if there are more benefits, whether the benefits apply for all companies involved with experimentation, or whether the benefits could be obtained through other means. Another benefit is the potential usage of continuous experimentation for software quality assurance. Continuous experimentation could support or even change the way quality assurance is done for software. Software change, for example, could only be deployed if key metrics are not degraded in the related change experiment. Thus, quality degradation could become quantifiable and measurable. Although some papers, like [100], mention the usage of continuous experimentation for software quality assurance.

6 Conclusions

This paper presents a systematic literature review of the current state of controlled experiments in continuous experimentation. Forward snowballing was applied on the selected paper sets of two previous mapping studies in the field. The 128 papers that were finally selected, were qualitatively analyzed using thematic and narrative synthesis.

The study found two constituents of a continuous experimentation framework (RQ1): an experimentation process and a supportive infrastructure. Based on experience reports that discuss failed experiments in the context of large-scale software development, the recommendation to practitioners is to apply one of the published processes, but also expand it by placing more emphasis on the ideation phase by making prototypes. As for the infrastructure, several studies discuss requirements for controlled experiments to ramp up the scale and speed of experimentation. Our recommendation for infrastructure is to consider the organizational aspects to ensure that, e.g., the necessary channels for communicating results are in place.

Ten themes of solutions (RQ2) were found that were applied in the various phases of controlled experimentation: data mining, metric specification, variants of controlled experiment design, quasi-experiments, automated controlled experimentation, variability management, continuous monitoring, improved statistical methods, and qualitative feedback. We have provided recommendations on what problem each solution theme solves for what context in the discussion.

Finally, the analysis of challenges (RQ3) and benefits (RQ4) of continuous experimentation revealed that only two papers focused explicitly on the benefits of experimentation. In contrast, multiple papers focused on challenges. The analysis identified six themes of challenges: cultural/organizational, business, technical, statistical, ethical, and domain-specific challenges. While the papers on challenges do outnumber the papers on benefits, there is no cause for concern, as the benefits to product quality are also mentioned in many papers as motivation to conduct the research. The challenges to experimentation also come with recommendations in the discussion on how to mitigate them.

As a final remark, we encourage practitioners to investigate the large body of highly industry-relevant research that exists for controlled experimentation in continuous experimentation and for researchers to follow the many remaining gaps in literature revealed within.

Acknowledgements

This work was partially supported by the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation and the Austrian Science Fund (FWF): I 4701-N.

Paper II

Continuous Experimentation Scenarios: A Case Study in e-Commerce

Rasmus Ros Elizabeth Bjarnason

Abstract

Controlled experiments on software variants enable e-commerce companies to increase sales by providing user-adapted functionality. Our goal is to understand how the context of experimentation influences tool support. We performed a case study at Apptus that develops algorithms for e-commerce. We investigated how the case company uses experiments through five semi-structured interviews. We identified four main scenarios of experimentation and found that there are stark differences in tool support for them. The scenarios illustrate that the aptness of tool support for experiments depend on four characteristics: (1) what the goal of the experiment is; *validate* or *optimize*, (2) whether the experiment is performed *internally* in the organization or *externally*, (3) whether decisions are taken *automatically* or *manually*, and finally (4) whether the experiment should be *repeated* or is a *singleton*. These insight can be used by practitioners with an interest in efficient experimentation and to form a basis for further research into a taxonomy of experiments for software.

Keywords A/B-testing · Continuous experimentation · Tools · Automation · Optimization · e-Commerce · Case study

1 Introduction

Web companies such as Microsoft [174] and Google [286] use continuous experimentation [106] to guide the development of new value-adding features according to user preferences. Users are exposed to different software variants in a controlled experiment using, e.g., A/B testing. Decisions are then made regarding which variant to release based on the users' responses to these variants. Based on this knowledge, new ideas are formed and experimented on. In this way, continuous experimentation can enable companies to meet the needs and preferences of their users, and when done with speed this is a key competitive factor [32]. E-commerce represents the frontier of continuous experimentation, based on numerous experience reports at, e.g., Amazon [138] and Microsoft [174]. We believe that there is much to learn about the state-of-the-art experimentation in e-commerce.

We performed a case study at a small business-to-business company called Apptus that develops a software product with algorithms for enabling automatic optimization of e-commerce solutions. Apptus is well versed in experimentation. They support their customers, i.e., e-commerce companies, in experimentation for their web shops and have access to data from multiple web shops. Apptus applies a rigorous experimentation process and have developed tools for automating experimentation. Our aim in studying Apptus is to learn more about how continuous experimentation is automated successfully within e-commerce. In particular, whether every experiment should be automated. Therefore we qualitatively characterize the contextual details around the experiments to see which roles and software products are involved, and what the degree of automation is. We detailed this as a research question, scoped within e-commerce: *In which scenarios are experiments used and what tools exist to support those?*

It is necessary here to clarify what we mean by automating experiments and tool support. We use the RIGHT model [106] to explain what parts of the continuous experimentation process is automated. The process encompasses the whole development process from defining hypotheses, to executing and analysing experiments, and finally to decision making. Some tools automate only the execution of experiments, e.g., PlanOut by Facebook [15] and the tools in use by Google [286] for running thousands of experiments in parallel. Recently, some tools (all of them applied to e-commerce) use large scale optimization methods that can experiment on hundreds of variables simultaneously. The tools are used to allow the software to adapt to multiple contexts such as personalisation to users or to handle different deployment settings. In this way, the decision making step of the process is automated. The tools have employed optimization methods such as genetic algorithms [208, 250] or multi-variate bandit optimization [138]. Bandit optimization is also in use at the case company [40].

2 Method and Case Company

We performed a case study to obtain insight into how continuous experimentation is applied within e-commerce, using guidelines by Runeson et al. [254]. We defined our research question (see Section 1) based on previous studies of controlled experiments for e-commerce [138, 174] and aided by our pre-knowledge. The first author's prior industry experience of e-commerce enabled us to design a focused investigation. We designed a semi-structured interview guide [245] covering: (1) role and background, (2) experimentation process, (3) planning, (4) analysis and knowledge building, (5) infrastructure, and (6) challenges.

We conducted our study at Apptus, a Swedish company which develops algorithms for e-commerce: recommendation system, search engine, ads display algorithms, etc. These software components are packaged as a single product that shares behavior data. The Apptus software product provides web shops with intelligence and can be considered “the brain” of these shops.

Apptus has a business-to-business relationship with their *customers* who operate e-commerce web shops. These customers usually have a business-to-customer relationship with the *consumers*. This study focuses primarily on Apptus' use of experimentation. The consumers are the source of hypotheses for these experiments. For Apptus, these hypotheses mainly impact the algorithms used in their software product, and are elicited through exploratory data analysis and experiments by developers and data scientists. Even though Apptus has no direct relationship with the consumers they have access to consumer data for their customers' web shops.

Apptus has around 40 employees of which around 20 are software engineers split over five cross-functional teams. Roles and their involvement in the experiment process are similar to those defined in the RIGHT model [106]. *Product owners* define experiments and set goals. The experiments and the connected metrics are designed by *data scientists*, while *software developers* and *quality assurance* implement and verify the functionality. This functionality is put into production by the *operations* and *release engineers*, that also execute and monitor the experiments. However, the case company does not have personnel with a dedicated quality assurance or release engineer role, instead they share these roles among themselves.

We used snowball sampling for selecting interviewees and interviewed five people, namely two data scientists, one software developer, one operations engineer, and one product owner. The interviews were conducted in Swedish for about one hour each during in Spring and Summer of 2017. The first author transcribed the audio recorded interviews word-for-word (40 pages of text) and a summary was sent to each interviewee for validation. The transcripts were analyzed using thematic coding in two steps, first using explorative descriptive codes (by first author) that were then grouped into themes (by both authors).

3 Results: Experimentation Scenarios

We identified four main scenarios of experimentation, namely (a) manual and (b) automated optimization, and (c) external and (d) internal validation of a new feature or product, see Figure 1. The scenarios were gathered from the interview material regarding experiments conducted and the tools developed at the case company. We now outline the aim of and roles involved in each scenario. For each scenario, we also present steps that the case company have taken to support automation for these, with the exception of external validation (Scenario c) where no automation was mentioned by our interviewees. The differences between the scenarios were commented on by an interviewed data scientist (translated from Swedish): *“Optimization is done because you have a lot of different components that work in different settings, and you do not have the time and ability to do that for all sites. [...] Use A/B tests when you have a hypothesis and you want to know something about the world. That is completely different. Both are really important.”*

3.1 Manual optimization scenario

When manually optimizing a web shop for, e.g. increased product sales, this is performed by marketers responsible for the web shop, see Figure 1a. In this scenario, we focus on the experimentation that uses the case company’s software product, but experimentation was said to be used elsewhere too, such as marketing experiments involving variations of e-mails in campaigns to users. A marketer holds a crucial role in e-commerce. They organize the day to day business of a web shop and perform activities such as creating sales campaigns that push certain consumer products, perform social media advertising, write blog posts, or also perform optimization experiments on the web store. As explained by the interviewees, marketers are not software developers and must rely on content management systems and tools for experiments. The case company provides experimentation functionality, but there are also many partially competing content management systems (e.g. Magento, Shopify, or EpiServer) and experimentation tools (e.g. Optimizely or Google Analytics).

Tool support: A/B testing for algorithms

The case company has developed a tool for aiding marketers in their experimentation supporting the manual optimization scenario. This tool compares variations in algorithmic functionality on a section of the web shop against each other. If the functionality of all variants in the experiment should return the same type of content, then no changes need to be done to the software used for the web shop. For example, comparing a *top sellers* al-

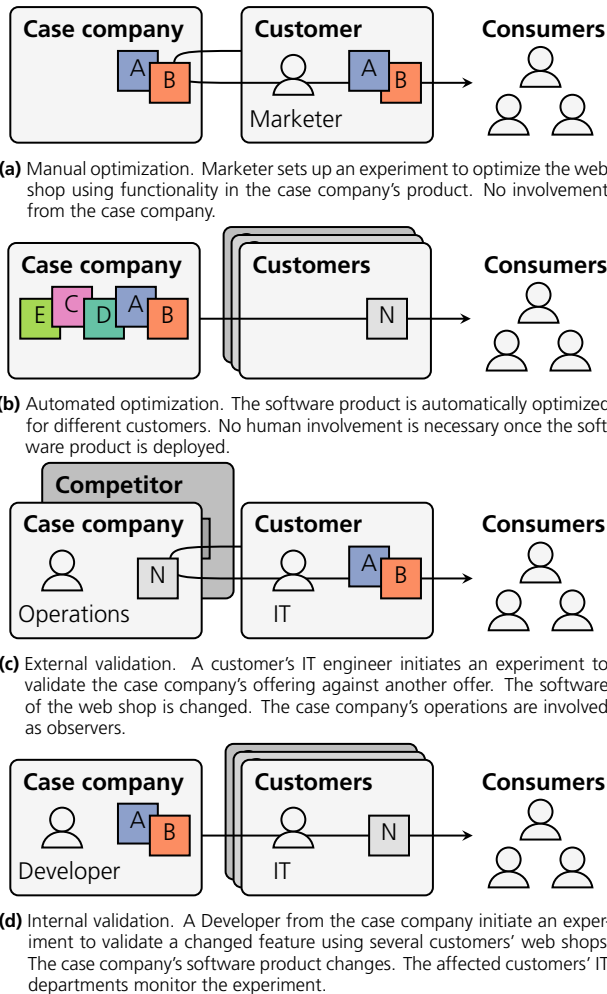


Figure 1: Experimentation scenarios. The large rectangles depict the companies and the roles involved, arrows indicate the initiator of the experiments, the squares represent software components, with multiples of them—marked A and B—indicate variations of the software and a grey box with N indicate no change.

gorithm to a *recently viewed* algorithm; they both display a list of consumer products. This can also be applied to tuning algorithm parameters (e.g., search ranking settings), product filters (such as removing adult content from recommendations), and to promote certain products in recommendations.

The developer mentioned that this tool in principle could be fully automated, so that the marketers were relieved of this work. However, the operations department from the case company has frequent contact with marketers and explained that the marketers appreciated the control and input to the algorithms it gave them. This was said to increase the trust that the marketers put into the case company's algorithms.

3.2 Automated optimization scenario

In this scenario, the software product automatically optimize a web shop towards a given metric, see Figure 1b. The case company's developers and/or data scientists develop the algorithms. The developers also specify the variations in the parameters involved with the automated experiments. Once set up, it does not require additional intervention. The product owner described that optimization is the core business of the case company, i.e., they support their customers in optimizing their web shops for increased sales, revenue, or profit. Since each web shop is different, the case company cannot directly apply results from experiments on one web shop to another web shop since these findings may not be generalizable. Also, the case company cannot themselves manually optimize every web shop due to the manpower this would require. Subsequently, the automated optimization scenario plays an important role for the case company.

From the interviews there were several examples of automated optimization that did not use an experimental methodology, i.e., they do not have different variants of the software product. As an example, ordering search results based on how relevant products are to the search query is a form of mathematical optimization, but it is not an experiment. Both kinds of algorithms improve by observing more data. The differentiator is that algorithms performing automated experiments internalise knowledge about the performance of variants through statistical inference. This is crucial, because it means that valid inference from the results can still be made in order to guide future software development.

Tool support: Interleaved recommendations with fully automated experimentation through bandit testing

The recommendation system in use at the case company [40] uses automated experimentation to optimize the product recommendations of web shops. The system aggregates recommendations from multiple sources and resolve which source works best depending on the circumstances it is used in. It is a large-scale bandit test that combines several product recommendations from various sources and presents them together in an interleaved fashion. Interleaving can be used to A/B test functionality that returns several results (such as recommender systems or search engines) by mixing recommendations evenly from either source.

3.3 External validation scenario

In this scenario, a customer's IT engineer evaluates and compares a new product offering from the case company with a previous or alternative implementation of the web shop, see Figure 1c. The IT engineer sets up the experiment and an operations engineer from the case company might be involved to ensure that the comparison is fair. The experiment is conducted on a small portion of the functionality, such as consumer product recommendations, but could in theory be a whole-site experiment. This type of experiment is usually only performed when new customers are acquired or lost to a competitor. The experiment can be done either on the initiative of the customers or by the case company. The performance of the algorithms is very crucial for the case company, and presumably for their competitors. As a consequence, the sales/marketing department of the case company is often eager to have such experiments performed, since this can provide evidence of their offering compared to their competitors. In some cases the case company is not notified about the experiment until after it has been conducted. This provides a strong incentive for the case company to have a product that is easily configured and integrated without mistakes.

Tool support

No tools were mentioned for this scenario.

3.4 Internal validation scenario

In this scenario, a developer and/or data scientist from the case company implement an improvement to an existing feature in their software product and conduct an experiment to validate this new software product variant, see Figure 1d. In this case, a previous version of the same feature typically serves as a control variant. Only the software product from the case company changes, but the IT department running the web shop is also involved in monitoring, and in the case of on-premise installations, deploying the software product. These experiments are fairly common. However, each development team within the case company usually conducts only one such experiment at a time. Overlapping experiments for internal validation [286] are not necessary for the case company due to having access to multiple web shops.

Tool support: Internal A/B test tools

The case company has several tools supporting the internal validation scenario. The developer mentioned that compared to the tools for manual optimization (Scenario a), the internal tools are more complex and highly configurable, and they are under constant change to suit their changing needs. These tools also support a much broader set of metrics compared to those that are used by their customers. For example, metrics that relate to the internal health of the software system.

4 Discussion and Conclusions

We have conducted a case study to understand tool support for continuous experimentation within e-commerce by investigating Apptus, a small company that develops algorithms for web shops. We found four scenarios of experimentation with varying tool support (see Figure 1), ranging from no tools for external validation (Scenario c) to fully automated decision-making for automated optimization (Scenario b).

We discern four main characteristics for the experimentation scenarios identified in our case study, namely (1) experiments are conducted to *optimize* settings or to *validate* a new feature; (2) experiments are *internal* or *external*, defined as whether the experimenter have access to and knowledge about the code base; (3) decision making based on experiments can be *automated* or *manual*; and (4) some experiments are *repeated* in multiple contexts or are *singletons* and run only once. Following this, the scenarios in Figure 1 can be further categorised. For example, in the manual optimization scenario (Scenario a) the experiment is *singleton* and both *internal* and *external* as it involves multiple software products where the marketer has deep knowledge of only one product, i.e. the web shop. Furthermore, both validation scenarios (scenarios c and d) contain manual experiments. The observed scenarios do not cover all possible combinations of characteristics. For example, we did not find any evidence of automated validation.

The different characteristics have varying need and feasibility of tool support. In the *optimization* scenarios, experiments are used to optimize software by tweaking algorithmic hyper-parameters and cosmetic user interface changes [138, 208]. The end goal of the *validation* scenarios can also be optimization, but the method to achieve this is not an experiment per se. For example, when introducing a new feature, the end goal can be to improve the user experience and an A/B test is conducted to validate this hypothesis. We conclude that the difference in the type of change and focus is the reason that automation of decisions is not needed for validation. We also observe different levels of tool support for the *internal* and *external* characteristics. Internal validation (Scenario d) has tool support

for configuring and deploying experiments. This is the case at the case company and in companies such as Google [286] and Microsoft [174]. For external validation (Scenario c) there was no tool support for automation. Our interpretation is that this is due to technical reasons, such as the lack of ability to directly change the software.

We have identified the following three threats to the validity of this study. (1) *Reliability*. The first author was previously employed at the case company, which poses a risk of bias. We mitigated this by the co-author's involvement and by letting another researcher review the study protocol. We also asked the interviewees to comment on coding transcript summaries and on this paper. (2) *External validity*. The extent to which the results of our case study are transferable beyond our specific case needs to be assessed by theoretical generalisation and comparing case characteristics. (3) *Construct validity*. In interview studies there is a risk that the questions are misunderstood. We mitigated this risk by selecting interviewees that were familiar with the subject of our study through their daily engagement in practically performing experiments. In addition, the first author's pre-knowledge of the company was used to design the interview questions thereby mitigating the risk of using misaligned terminology.

Through our study, we report on real world use of tool support for experiments and conclude that it is possible to improve the scale and speed of experimentation through automation. However, the tools are not general and the specifics depend on multi-faceted factors such as the business model and the software ecosystem. Future work include the use of tools for experiments in other domains. With sufficient empirical data a taxonomy of experiments can be constructed.

Acknowledgements

We want to thank the interviewees at Apptus. We also thank Professor Serge Demeyer and Professor Per Runeson for helpful reviews of our manuscript. This work was partially supported by the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation.

Paper III

The FACE Theory for Factors at Play in Continuous Experimentation

Rasmus Ros Elizabeth Bjarnason Per Runeson

Abstract

Context Continuous experimentation (CE) is used by many companies with user-intensive products to improve their software with user data. Some companies deliberately adopt an experiment-driven approach to software development while some companies use experimentation in a more ad-hoc fashion.

Objective The goal of the study is to find factors that explain the variations in experiment utility and efficacy between different companies.

Method We conducted a multi-case study of 12 case companies and 27 interviewees at companies involved with experimentation and built a theory on CE based on that data.

Results We introduce a theory of Factors Affecting Continuous Experimentation (FACE) theory, which includes the three factors: (1) investments in data infrastructure, (2) user problem complexity, and (3) incentive structures for experimentation.

Conclusions We also provide guidelines and recommendations on how to achieve state-of-the-art CE. These guidelines are based on FACE and can support practitioners in improving CE even for companies with contextual factors that have a negative effect on their ability to experiment.

Keywords Continuous experimentation · Data-driven development · A/B testing · User experience research · Theory building · Multi-case study · Empirical research

1 Introduction

Continuous experimentation (CE) is an experiment-driven software engineering approach, where assumptions about product features and requirements are continuously tested with experiments, to reduce the risk of wasting development resources on requirements of little or no value to the users. In CE, software changes are compared to an old version in an experiment (such as an A/B test) with real users and only changes with a positive effect on usage are permanently released. Initially, this approach was primarily applied by large internet-facing software engineering companies such as Microsoft [176, 172], Google [286], and Facebook [107] that apply an agile and continuous approach to software engineering. Lately, also more traditional software engineering organizations such as companies in the business-to-business domain [242] are recognising the value of moving towards an experiment-driven development model that provides the ability to evaluate product assumptions through continuous user feedback [106].

Performing CE is not an easy task and one that requires companies to adapt their processes, structures, technical infrastructures, and culture [105, 320]. Several researchers report on the benefits of adopting CE and a high-level model on how CE is structured (the RIGHT model) has been proposed [106]. Despite this, not all companies are able to adopt CE, even though they might have sufficient user data available. We pose that this is due to the complex nature of software engineering and that many different factors and processes are at play. Some of these factors have been identified in previous research, such as having a service-oriented offering [265] or one with cyber-physical systems [33, 122] and whether the target users are consumers (B2C) or other businesses (B2B) [242, 317]. Also, only some companies are able to conduct experiments that affect business value [265]. However, there is a lack of overarching theory that can explain what factors explain the different efficacy of CE. For this reason, we wanted to investigate if there are any common patterns that can explain the varying utility, challenges, and benefits [11] experienced by companies when applying CE.

We have performed a multi-case study on 12 companies to explore CE practices and contexts [63, 230] for a range of companies, and constructed an empirically-based theory through iterative and systematic analysis of this data. Our empirical data consists of 27 semi-structured interviews with various roles relevant to experimentation, such as software developers, quality assurance, data scientists, and product owners. We have previously published two initial findings based on part of the interview material. First, a paper about scenarios in which experiments are used [247] which give an indication that experimentation context matters. Second, a paper on the role of business models with product-led growth [246]. These findings are synthesized into a small part of the contribution of this paper, and seven additional case companies are added.

In this paper, we present a theory of Factors Affecting Continuous Experimentation (FACE) based on empirical data from the 12 companies. FACE considers socio-technical factors in organizational contexts surrounding CE and observed relationships between factors. According to FACE, there are three main factors that influence how well an organization can expect to conduct CE. (1) There are requirements on *CE processes and infrastructure*, in particular, the data infrastructure that companies have for conducting telemetry and analysis of results. (2) The complexity of the *problem* that the software solves for its users limit experiment applicability. For example, a software system that integrates many other systems is highly complex and difficult to make changes in, thereby hindering CE. (3) FACE also highlights the role of *business models* in both providing incentives towards product improvement and for making relevant metrics available for experimentation or not.

Finally, we derive guidelines [278] from FACE that show what state-of-the-art experimentation is like and provide recommendations on how to strive towards it. In summary, companies with state-of-the-art CE utilize user data to prioritize what to experiment on, have dedicated teams for supporting experimentation, and apply experimentation throughout the organization. Though, our results show that getting started with experimentation is technically easy, increasing experimentation throughput is very hard. So companies in the process of starting with CE should focus on gaining practical knowledge of experiments which will be useful when scaling up experimentation.

The rest of this paper is structured as follows. The background and related work on CE and theory building are presented in Section 2, and our research method is outlined in Section 3. FACE is presented in Section 4 and the empirical underpinnings and explanations are provided in Section 5. The findings are discussed in Section 6. In Section 7, guidelines are derived from FACE on how to adapt the CE practice at a particular organization. Finally, we conclude the study in Section 8.

2 Background and Related Work

Companies in many different sectors have adopted Continuous Experimentation (CE) [9, 249], where features are evaluated through user feedback. Prototypes of a feature or product can be quickly validated with users before a costly implementation is finalized and released to all users. After implementation, the change in software can be subjected to a controlled experiment (such as an A/B test) where a comparison can be made with and without the new change. Only changes that have a positive impact on user feedback are accepted. The results of an experiment might beget further questions, especially for negative results to figure out what went wrong. Thus, experiments are usually executed in a sequence, which

is why the practice is coined *continuous experimentation*. Both prototyping pre-deployment experiments and controlled post-deployment experiments are of interest in this study. The topics covered in this section include software business model and how it relates to CE, an overview and related work of CE, and theory building in software engineering.

2.1 Software business models

CE provides a way to measure the value that software development brings to users and business. As such, studying CE entails understanding how that value is delivered. In this study, we use business models and business strategy as a lens to structure our analysis of that value delivery. A *business strategy* in the management field is a long-term vision for a company [154]. A business model is a concrete plan to execute that vision. The term *business model* is narrowly used in industry to refer to how a company collects revenue [293]. In this paper, the broader definition by Osterwalder [226] is used: “*A business model describes the rationale of how an organization creates, delivers, and captures value*”.

Recently, in the business model innovation field, prototype experiments have been studied as a method to find a combination of a working business model and product [43, 276, 311]. This contrasts with the way experimentation is viewed in CE research, where the focus is more on product improvement [106], and changing the business model might be considered out of scope for daily software engineering work. In addition, successfully implementing changes in an established business model is notoriously hard and risky [59, 60]. As such, business models are mainly used to describe the context of companies in this study, not as a subject of experimentation.

There have been several attempts to describe commonalities in software engineering business models with frameworks [238, 266]. For example, Rajala et al. [238] include aspects of product strategy, revenue logic, distribution model, and service and implementation model. The business model canvas [225] is probably the most popular framework to describe business models succinctly. We use an adaption of this framework as described in the next subsection.

2.1.1 Lean startup

Lean startup is a methodology, originating in industry from Ries [240], that applies lean manufacturing principles [178] to entrepreneurship in general. Lean startup has also been studied in a software engineering context [26, 35, 106]. The idea is to conduct product development in short cycles to obtain feedback on whether a proposed business model is viable as early as possible. Ries calls it the build-measure-learn cycle, where each cycle consist of a business hypothesis and an experiment to verify the hypothesis. As such, CE and lean startup has a clear connection.

The experimentation that lean startup calls for is done through prototyping a solution and testing it on a limited set of customers [26, 132, 294]. The goal of this experimentation is to find a minimum viable product (MVP), which is the smallest set of features that solve the users' problems. It cannot give accurate numbers that can be used to compare different solutions, as is the case with A/B testing. As such, the prototyping experiments have low fidelity but have a low cost compared to A/B tests that must be executed on finalized software in a production environment.

Maurya proposes an adaptation of the business model canvas to suit lean startup needs, called the *lean canvas* [201]. The lean canvas is divided into a product and a market part. The product part contains: the problem-solution pair that the product addresses, what key metrics are measured, and what the cost structure is for acquiring customers, developing code, operations, etc. The market part contains what the unfair advantage is, such that the product cannot be easily copied, what the channels to customers are, what the target customers are, and the revenue streams. The two parts are tied together with a value proposition message.

Maurya [201] also describes the three phases of a startup that (1) start with finding a working problem-solution pair, (2) then the product should have a market fit, and finally (3) once this is validated can growth be the focus. Experiments are used differently at these stages where the initial focus is prototyping and later on controlled experiments can tune the product for product-market fit and growth.

2.1.2 Product-led growth and growth engineering

A specific archetype of business models has recently been popularized in industry, under the name of *product-led growth* [18]. A business model that has a product-led growth relies on the product itself to acquire new end-users rather than on the direct sales & marketing activities (e.g. advertisement or cold calling) of the sales-led business models. The purpose is to have an offering that can scale to high levels of demand. Furthermore, a new role has also been introduced to business with product-led growth [162, 288], called growth

marketers, growth engineers, or growth hackers. These roles are hybrids between marketers and software engineers that work data-driven with experimentation and analysis to propel a company's customer acquisition growth. Two of the companies in our study have employees with such titles.

A more precise definition of product-led growth—as the term is used in industry—has not been found. Instead the following characteristics are derived from Bartlett [18]:

- the software development organization elicits requirements in order to meet market needs;
- there is no customer specific development in order to ensure software development is directed towards improving the product for all users;
- the channels to acquire customers are scalable to many customers and are often organic (i.e. word of mouth instead of direct sales);
- the primary source of revenue is through product sales or subscriptions.

The last of the above characteristic, regarding the source of revenue, has to do with the licensing model that software is sold under. According to Bartlett [18], product-led growth is associated with *freemium*. A freemium product [217] is available both for free and as a paid premium version. The premium version might, for example, have more features or offer improved customer support. As such, freemium encourages growth by allowing more customers to use the product and spread the word.

2.2 Continuous experimentation

CE has been studied from many different perspectives [247]. In software engineering venues, the topics have been varied, e.g., designs of specialized tools for optimizing experiments [248, 262] and descriptions of the CE process in use at various companies [31, 106]. Although software engineering is the focus of this study, there has also been considerable practitioner focused research on CE in data science and user experience research venues. In the data science field, the seminal paper by Kohavi et al. [172] provides a practical guide to starting with controlled experiments on software in a production environment. In user experience research [258, 267], experimentation has a less prominent role among other competing methods such as user observations, card sorting, or interviews.

2.2.1 Process and infrastructure models

The infrastructure and process of how several different companies conduct CE have been described with reference models and experience reports [31, 107, 176]. The models are conceptual generalizations based on observations of CE in industry, thus they cannot be used to explain organization efficacy of CE which is the purpose of FACE. The RIGHT model by Fagerholm et al. [106] contains a description of the process and infrastructure needs for conducting continuous experimentation based on a multi-case study. The earlier HYPEX model by Holmström Olsson and Bosch [221] has similar goal but is less comprehensive. These reports and models served as a starting point for the questions in the interview guide in this study, in particular the RIGHT model.

The process model in RIGHT is inspired by the build-measure-learn cycle of lean start-up [240]. Fagerholm et al. use the concepts of a minimum viable feature (MVF) to bridge the theoretical gap between prototyping experiments and controlled experiments, and so the model considers both types of experiments. There are five main phases of the CE process in RIGHT. (1) In the *ideation* phase hypotheses are elicited and prioritized and a change to the software is proposed. (2) *Implementation* of the minimum change that tests the hypothesis follows. (3) Then, a suitable *experiment design* and a criterion for success is selected (a metric in the case of a controlled experiment). (4) *Execution* involves deploying the product into production and monitoring the experiment. Finally, (5) an *analysis* and decision is made whether the results are satisfactory, if not the process restarts.

The technical infrastructure needs in RIGHT include tools for managing experiments and analytics, instrumentation in the product, and a continuous delivery pipeline. These tools are often referred to as an *experimentation platform* when considered as a whole [130, 172]. The organizational infrastructure includes roles involved with experimentation, of which there are many, since the experimentation phases cover the whole software engineering process. The necessary roles are according to Fagerholm et al. [106]: Business Analysts and Product Owners elicit hypotheses and maintain an experimentation road map; Data Scientists design, execute, and analyze experiments; Software Developers and Quality Assurance develop and verify the software; and Operations Engineers and Release Engineers deploy and deliver the software.

2.2.2 Factors affecting continuous experimentation

We are aware of one other attempt at analysis of how effective experimentation is for various companies, albeit from a startup perspective, by Melagati et al. [206]. They studied factors affecting CE in terms of enablers and inhibitors of experimentation at early-stage startups. Many of the identified inhibitors point to a lack of resources to conduct experiments, which is more pressing for startups due to a general lack of resources. Also, the research only considers pre-deployment prototype experiments, which is much less technically demanding.

There are also many studies that describe or report experiences about conducting CE for various circumstances. We have identified five such clusters of papers with domain specific challenges in a systematic literature review [11]. Two of the clusters relate to factors that might influence the gains with CE: business-to-business (B2B) and cyber-physical systems. The challenges in the remaining three clusters are either overlapping (mobile and cyber-physical systems) or potentially solved by statistical solutions (e-commerce and social media).

The challenges with CE in the *business-to-business* (B2B) cluster are many [242, 247, 317]. B2B companies are usually involved with their customers' software engineering or IT departments, which causes issues with control of software deployment or with access to end-user data. Both of these are necessary and require constant collaboration efforts to resolve. Furthermore, the incentives to improve the software product in terms of end-user experience might not be there. For example, if the company in question generates revenue per project instead of through the value delivered to users. These issues might not be faced to the same degree by companies with a business-to-consumer (B2C) business model. Whether a company is B2B or B2C is not an explicit part of FACE; these challenges are instead explained by other constructs in FACE (business model and problem complexity).

As for the *cyber-physical systems* [33, 122, 197, 222] cluster, the focus of the research is on suggesting and describing the required infrastructure to enable continuous experimentation, such as continuous delivery of software and how telemetry can be implemented. Without this, experimentation is not possible. The body of work on these domains is still in the early stages and no such companies are included in this study.

2.2.3 Previous work

We have two prior publications from a research project started in year 2018 that uses five of the 12 cases that this study is based on [247, 246]. The findings from these papers are synthesized and expanded in FACE.

The first paper [247] was a single case study on a B2B company that develops behavior algorithms and used experimentation in four different scenarios. (1) To verify that changes in their algorithms are beneficial. (2) To help their customers use their algorithms correctly. (3) To prove that their algorithms outperform their competitors'. Finally, (4) as a black box optimization method to tune the algorithms automatically [41]. The degree of tool support was different between the scenarios, ranging from fully automated to no support. The scenarios give an early indication on how experimentation differs depending on their purpose.

The second paper [246] used five of the 12 cases in a comparative case study. The study was about the role that a company's business model plays in relation to CE. The differences attributed to business models were sufficiently explained by whether the business model had product-led growth or not (corresponding to one of the three factors derived from FACE). Four drivers of a product-led growth focus were identified as affecting experimentation. (1) Development and sales & marketing worked more closely together at the product-led cases, with mutual benefits. (2) The prioritization process used data to a higher degree to inform development. (3) What features were included were based on market needs rather than by customer requests, thereby decreasing excessive feature bloat. Finally, (4) the availability of metrics relevant to business was higher in the product-led cases.

2.3 Theory building

Theories provide a means of structuring and conveying knowledge in a condensed form. Theories can be of different types, some can provide explanations of phenomena or predict the outcome, e.g., of applying a certain practice. Theories that describe a particular aspect of software engineering are gaining traction (see, e.g., Rodriguez et al. [243] or Munir et al. [213]), presumably due to their usefulness in supporting research design and on improving software engineering practice. According to Stol and Fitzgerald [278], it is preferable to base practitioner guidelines on theories such that they are underpinned with theoretical knowledge on why they hold.

We used the guidelines by Sjøberg et al. [274] to build FACE. In Sjøberg et al., the theory is constructed based on a generalization of multiple cases observed in real world. There are other theory building approaches [304], such as basing them on existing theory from other fields or grounded theory [124, 281], which is used when the researchers want no preconceived notions about how the data should be interpreted. A theory, as defined by Sjøberg et al. [274], consists of *constructs* that the theory makes statements about in the form of *propositions*, which are relations between the constructs. The theory also contains explanations about why the propositions hold and a scope that the theory is valid under.

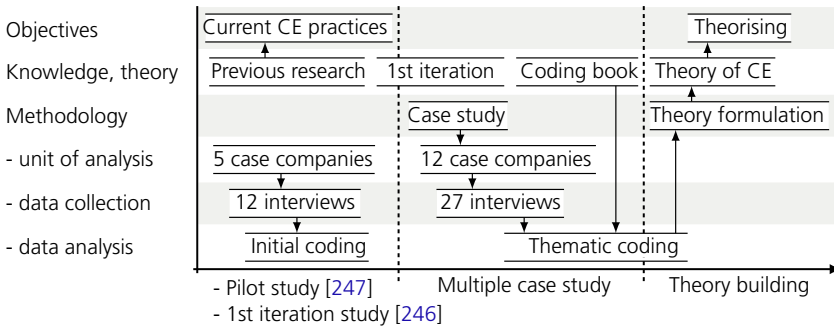


Figure 1: An overview of the research method through which FACE was constructed based on a multi-case study. Two previous studies (to the left) were used as a starting point for the multi-case study.

3 Method

This paper presents a *multi-case study* of companies applying CE and a *theory building* process aimed at providing a generalized description of factors affecting a company’s ability to obtain gains from CE. The resulting theory was inducted through iterative analysis of the interview data. An overview of our research method is provided in Figure 1. The empirical data in our study is from case companies and we use the guidelines by Runeson et al. [253] for case study research, covering the interview process and thematic coding. The theory was then created through the codes and themes, using the process for building theories by Sjøberg et al. [274], and guidelines were derived as motivated by Stol and Fitzgerald [278]. The theory is based on data from 12 case companies, denoted A–L, see Section 3.3.

3.1 Multi-case study with interviews

We performed *semi-structured interviews* in order to gain insights into what and how contextual factors affect an organization’s ability to perform CE, and thereby provide a rich empirical basis for our theory building process. We designed an *interview guide* based on previous related research [106, 172, 221], *sampled and recruited companies and practitioners* to include in our study, performed the *interviews* of these using an interview guide, and *analyzed* the interview data using a code book. The first author led the interview study, including the design of the interview guide and the code book, recruiting of the interviewees, performing and analysing all of the interviews. The second and third authors reviewed and provided feedback on the research design and the research artefacts, provided feedback on the interview guide, participated in two interviews each, and performed independent coding of one interview to improve the coding process.

This study is part of a three year research project with two previous publications (see Section 2.2.3). The project was started with an exploratory pilot case study on a company with established experimentation. Some of the findings from that pilot study has been previously published [247]. An initial version of the interview guide and code book used in this study was created for this pilot study. The pilot study was followed up with a first iteration of this study, on the role of business models in CE [246]. The first iteration study was conducted after all interviews were conducted, but included only five of the 12 cases.

An *interview guide* was used to support the interviews and to ensure that all relevant aspects were covered in each interview. We designed the guide based on our knowledge of the area and of previous research on CE, in order to cover as many relevant aspects as possible, and thereby further enhance the richness of the resulting empirical data. The descriptive models by Fagerholm et al. [106] and Holmström and Bosch [221] were used as the theoretical basis to derive the interview questions. Additional descriptions of experimentation from experience reports [176, 31] were used to ensure that the questions covered all aspects of the experimentation process and infrastructure, and the context around experimentation in terms of the organization and business. The interview guide was designed iteratively by the authors with small updates after each initial interview. The last 15 interviews used the final version of the interview guide, which version is provided in Appendix A. The interview guide consists of five parts: interviewee information, case context, experimentation process, experiment details, and holistic experimentation view. The guide was designed with probes such that it could be adapted to the interviewee's background and role.

We *sampled and selected* case companies through a mix of convenience sampling and snowballing guided by the aim of our study. We selected companies that either apply CE or have plans to start applying CE. In three of the cases we selected a branch of the company as the case (cases D, I and G). Candidate companies were identified through searching on LinkedIn and Google for job listings for data scientists where A/B testing was mentioned, through personal contacts, and through asking the interviewees (i.e. snowballing) for other companies involved with continuous experimentation. As such, the cases were not restricted by geographical location. Our aim was to interview practitioners with insights and experience of applying CE, and primarily in the roles with heavy experimentation involvement: software developers, product owners, and data scientist. We applied snowball sampling within the companies by asking for additional interviewees within the companies during the initial interviews. The managers closest to experimentation were contacted via e-mail with information about the study and a request to perform interviews. In total 35 companies were contacted of which 12 were included in our study. Of the remaining, 11 did not respond and 13 were not applicable. A description of each included case company is provided in Section 3.3.

The *semi-structured interviews* were held during an initial 3 month period for the pilot study, which was then continued a year and a half later for another year. Interviewees were selected from the case until a complete picture of how and why the case used experimentation was achieved. Albeit in some cases (B and E) the process was cut short due to the subsequent interview prospects not wanting to be interviewed. Each interview was approximately 60-120 minutes long and was held as an open conversation in-line with the interview guide. The interviews were audio recorded after permission for this was granted by the interviewees. The majority of the interviews were performed at the companies' premises. For six companies, this was not feasible due to their location, in which case the interview were held on-line via Zoom or Skype. After the interviews, the audio recordings were transcribed word-by-word into 231 pages. Some of the interviews were done in Swedish and quotes from those interviews were translated to English.

Thematic coding was performed on the interview transcripts using the pre-defined codes of our code book. The code book was defined based on previous knowledge and insights into CE, from primarily three core publications [31, 106, 172]. It was iteratively refined throughout the coding process by discussing these codes within the group of authors. The codes were added per paragraph of transcribed text, and each paragraph could have multiple codes. The codes were clustered into 11 themes according to thematic analysis. The codes cover areas such as the company context, management of data, the business model, product, and experimentation process for the company, see Appendix B. After having coded the full set of transcripts, cross-case analysis was performed to identify factors and patterns common to several cases. The theory was gradually inducted as this cross-case analysis matured, see below.

3.2 Theory building

We have built the theory by analysing our empirical data and gradually defining constructs, propositions, the scope and explanations for the theory, based on guidelines provided by Sjøberg et al. [274]. The final step of theory building, namely testing our theory, remains as future work. In this article, we provide an initial validation of our theory by characterizing each of our case companies based on the theory. In this way, we illustrate that the theory is useful for categorising and explaining an organization's ability to perform continuous experimentation.

The *constructs and propositions* of FACE represent the factors that affect CE and the relationships between these factors, relevant to CE. The constructs were identified through analysis of the thematically coded interview data. The themes from thematic analysis formed the first iteration of the constructs of the theory. An orthogonal coding step was carried out afterwards to find relations between constructs, which formed the propositions; using each theme-pair as a code. The constructs and the propositions were then gradually refined and

adjusted through discussion within the team of authors, to provide a clear and concise description of the factors that affect a company's ability to draw benefits from CE. Describing the relationships between the constructs through visualisation (see Figure 2) and written definitions (see Sections 4.2 and 4.3) facilitated this inter-author discussion. The supporting evidence in the material was used as a basis for the discussion. In total, 10 iterations of the theory were discussed in this way.

The *scope* and *explanations* of FACE were identified based on our empirical data. The *scope* of our theory was defined through considering the common characteristics of our case companies, and thus the type of organizations that our theory may be applicable to. Explanations for our theory are provided as part of the definition and description of each construct and proposition.

The *empirical underpinning* of FACE provides a motivation for the concepts of our theory grounded in the empirical data, thereby illustrating the empirical foundation for the constructs and propositions. In Section 5, for each proposition in the theory supporting evidence in the material is laid out along with expanded explanations. The empirical underpinning in terms of the constructs and propositions provides an *initial validation* of FACE, and illustrates its utility and explanatory power. As such, the empirical data is used both as a source and validation, by having the data used at the detailed level to construct the theory and then at the holistic level to describe our set of companies.

The *guidelines* were derived from the theory after its construction, as recommended by Stol and Fitzgerald [278]. The guidelines consist of two parts. First, a description of state-of-the-art practice which are sourced from the interview material either by direct statements or inferred observations. Second, actions that companies can take on how to adapt to specific factors which was done by systematically matching recommendations from the interview materials to the factors in the theory. The guidelines are aimed at practitioners for them to take action from the insights the theory provides. The guidelines are presented in Section 7.

3.3 Case companies

The twelve case companies in the study differ on many attributes such as size, product domain, business model and CE practices. Our case companies range from small to huge multi-national companies. Some of the companies work extensively with experimentation while some do next to no experimentation at all.

Table 1 contains an overview of the business models and states of experimentation practices for the 12 case companies. Each case company is ranked according to how much *expertise* they have with experimentation and the *extent* that they conduct experiments (both frequency of experimentation and on what parts of a product or service they can do experi-

ment on). The business model is given a brief summary, primarily describing their offering and what customers it targets. Under the business model column, *direct sales* refer to having a business model where licenses to the product or service are sold by contacting customers directly.

The 27 interviewees at the case companies are described in Table 2, which show their code corresponding to the case company and role name. Two additional roles have been observed in the interviews in extension to the ones presented in RIGHT [106] (see Section 2.2), namely, the *User Researcher* and *UX Designer*. They have similar responsibilities as data scientist and software engineers have, respectively, when it comes to experimentation. That is, a UX designer or software engineer comes up with a change in user experience or software, and a user researcher or data scientist analyzes the results. The user researchers in the interviews used primarily prototyping qualitative experiments. The actual titles differ significantly from their assigned role and the titles in use include, e.g., head of growth, software engineer, head of customer success, growth engineer, head of research, etc.

3.3.1 Case A: E-commerce algorithms

This case company offers an e-commerce platform that is sold to companies (B2B). The platform consists of various algorithms for ranking products and an administration interface. While the algorithms target end-consumers, the administration interface targets managers at the e-commerce companies. The algorithms provide value for the end-consumers by increasing the relevance of the product they see on the web shops. The case company is fairly small with about 50 employees and was established 20 years ago. The company's business model is to sell usage licenses to other companies, they have salespersons working with direct sales as their only source of revenue. They have experienced several periods of growth after which they have had to scale down when a big sale fails. They conduct a medium amount of experimentation but only on the software that targets their end-user consumers, not on the administrative interface. They only do quantitative experiments since the software they experiment on does not have a graphical interface. They also assist their business customers in their experimentation.

3.3.2 Case B: Local search service

Case B offers a search engine service for search within a local region, a.k.a., yellow pages. The search engine is free to use and the major source of revenue is companies buying promotion in the search result rankings and visual presentation. The case company has a large sales team that work with direct sales by calling potential business customers. The company is about 20 years old and has stayed stable for some years at about 50 employees. The

Table 1: Overview of our 12 case companies, containing business model, size and age (rounded to nearest 5 year), and state of experimentation. Small companies have less than 50 employees, medium have less than 250, large companies have less than 1000 employees, and huge more than 1000 employees. Experimentation expertise and extent is estimated based on our interviews using the ordinal scale: low, medium, and high.

Case	Business Model			Experimentation		
	Summary	Type	Size	Age	Expertise	Extent
A E-commerce algorithms	Product with direct sales	B2B	Small	20	High	Medium
B Local search	Service selling placement to B2B customers and serving ads to B2C	B2X	Small	15	Medium	Low
C E-commerce consultants	Consulting on a per project basis	B2B	Medium	20	Low	Low
D Video streaming	Freemium service with premium features	B2C	Medium	10	High	Full
E Web shop	Web shop for subscription to physical goods	B2C	Huge	10	Medium	Medium
F Customer relations	Product and service with direct sales	B2B	Small	30	Low	Low
G Engineering tools	Freemium service with premium for larger teams	B2B	Huge	20	High	Full
H Web shop	Retailer with small web shop	B2B	Large	20	Low	Low
I Web shop	Web shop and retail	B2C	Huge	20	High	Medium
J Product information	Service selling customer information and leads to B2B and free for B2C	B2B	Medium	10	Medium	Low
K Business intelligence	Product with direct sales	B2X	Large	30	Medium	Low
L Employee management	Service with direct sales	B2B	Small	20	Low	Medium

company was recently acquired by a larger business group and additional products were integrated with the local search service, such as a ticket booking service. Experimentation is not frequent at the case company and is only used to verify large changes to the search engine ranking algorithm with quantitative data.

3.3.3 Case C: E-commerce consultants

Case company C is a consultancy firm that develops and maintains web shops for other companies with relatively large demands on traffic volume and/or product catalogue. The work includes a lot of integrating various software systems, e.g., product information management (PIM), content management systems (CMS), payment gateway systems, search or

Table 2: Overview of the 27 interviewees at the case companies. The codes of the interviewees correspond to the case they belong to. The role is assigned by the authors to best describe their primary role(s) in experimentation—not their title.

Code	Role(s)	Code	Role(s)	Code	Role(s)
A1	Software Developer and Data Scientist	D2	User Researcher and Software Developer	I2	Product Owner
A2	Product Owner and Data Scientist	E1	Data Scientist and Software Developer	I3	Data Scientist
A3	Software Developer and Release Engineer	F1	Product Owner and Business Analyst	I4	Business Analyst
A4	Product Owner and Business Analyst	F2	Product Owner and Software Developer	J1	Product Owner
A5	Operations Engineer	G1	Quality Assurance	J2	Software Developer and Data Scientist
B1	Software Developer and Data Scientist	G2	Product Owner	K1	Product Owner and UX Designer
C1	Software Developer	G3	Data Scientist	K2	User Researcher
C2	Software Developer and Quality Assurance	H1	Software Developer	L1	UX Designer
D1	Data Scientist and Business Analyst	I1	Software Developer	L2	User Researcher

recommendation engines, etc. These software systems each come with their own administrative tools and the case company helps their customers to use these tools. They have built their own software to optimize and support the process of building the web shops, but the business model is to sell consulting hours in a per project basis. The company has existed for more than 20 years and currently have about 100 employees. They are experiencing steady growth in number of employees. The company does not conduct any experimentation on their own software but have assisted their customers in conducting quantitative experiments on their web shops.

3.3.4 Case D: Video sharing

Case company D develop and sell a video sharing platform where users can record and edit videos for marketing purposes. The company has been operating for 10 years, and has about 200 employees of which about 30 people are in the software development department, distributed over four teams. They recently pivoted in the business model by adding a new product that is targeted at individuals; aimed at consumers and smaller companies that wish to market themselves. This new product is the defining boundary of case D. Prior to the pivot their customers have mainly been other businesses (B2B), which they have reached through a fairly big sales department with direct sales. The new product has an entirely separate development department and no sales persons involvement. The pivot enabled

the use of experimentation at the company. The new B2C product is offered under a freemium license, where the free version is offered with limited video uploading capacity, and the paid version offers additional features. At the B2B part of the company they do no experimentation. In the B2C team they conduct extensive experimentation and have various specialized roles for supporting experimentation and related activities, viz., data scientists and data engineers for quantitative experiments and user researchers for qualitative experiments.

3.3.5 Case E: Web shop

This case company offers a web shop with a subscription service to physical goods. The web shop drives sales through their online presence only and does not have any retail stores. The company was founded less than 10 years ago and has experienced rapid growth in revenue and number of employees. They were about 3 000 employees at the time of the interview, most of them work with delivering the physical goods. The company's IT department accounted for about 250 employees. In addition to the web shop, the IT department also operates inhouse software systems for handling logistics, marketing, etc. The interviewee at the company was involved with optimizing multiple of these products at the company. There is a drive from the management to be data-driven and experimentation is encouraged on a strategic level. However, this is hindered in practice by chaotic management, caused by the rapid expansion.

3.3.6 Case F: Customer relations product

The next case sells a customer relations product (and service) to other businesses. The product is highly customizable and each new customer gives rise to a new integration project. The integration includes adapting to another company's data model and internal software systems, e.g., for human resources. The company is 30 years old and has 200 employees, 35 of which are in software development, divided in three teams. Sales is a large part of the company, and all sales are done through direct sales. Half of the company is committed to pre-sales and operations that support customers before and after sales due to the complex integration. No free version or trial of the product is available. They have conducted a few prototyping qualitative experiments on recent new features, but not on all developments. The company is aware of quantitative experimentation but has not conducted any, with no concrete plans to get started with experimentation soon. However, they have some of the technological pre-requisites to conduct post-deployment experimentation in place already—they make use of feature flags to verify new developments at specific customers, but not in a systematic way.

3.3.7 Case G: Software engineering tools

Case G is a huge international company with multiple products that focus on supporting the software engineering process. The main products are a project tracker, issue tracker, and a team collaboration platform. The company has existed for roughly two decades and today have more than 4 000 employees. The majority of these employees are within IT. Each of the company's products has its own development organization. The company advocates agile software development. Notably the company has no large sales department and do not use direct sales at all, despite their focus on business to business (B2B). The company conducts huge amounts of experiments on all aspects of their products—both on new and old products. They make use of both quantitative and qualitative experiments. A specialised team supports product teams in applying an experimentation approach. Since there are multiple teams involved with experimentation on multiple products, the company has an extensive experimentation platform. The company also has a formal process for conducting experimentation developed by the experimentation team.

3.3.8 Case H: Web shop

This case company offers a web shop for business customers only. The company has over 2 000 employees, of which most are employed at retail stores. The company used case company C to develop their web shop and also has their own small IT department that manages and improves the web shop. The company has conducted ad hoc experiments on occasion, they use off-the-shelf experimentation tools (Google Analytics) for supporting their experimentation. The biggest limiting factor of their experimentation is the lack of human resources at the IT department.

3.3.9 Case I: Web shop

The next case is a huge international conglomerate. The company has a long and rich history in retail and have operated a web shop for over 20 years. The web shop is becoming increasingly important to the company. The company has about 200 000 employees of which about 5 000 are employed in the IT organization. The company have many products for managing their logistics, etc., but in this study we focus on their web shop product only. Several cross-functional development teams are responsible for various parts of the web shop (such as the recommender engine). Not all parts of the web shop are experimented on, but the trend is moving towards more experimentation work. There are multiple teams involved with experimentation at the company, some of them have their specialized experimentation infrastructure but most teams use a centralized experimentation platform and contact a data science team for help with experimentation.

3.3.10 Case J: Product information platform

Case J offers a service for product information within the building industry that is free to use. The company was established less than 10 years ago. The source of revenue come from customers that want to know who accessed their product's information in order to obtain sales leads. The case company describes their business model as being a middle man that sells information. They have a sales department that works with direct sales and also relies on organic growth based on their free users. There are about 200 employees at the case company, half of which work within the IT organization. Quantitative experiments are new at the company and they have a small and dedicated team with two employees for conducting such experiments. They do not make use any of qualitative methods. The company expect that the other software engineering teams will also start with experimentation soon, but for now they do not experiment on all parts of their software.

3.3.11 Case K: Business intelligence

This company offers several advanced business intelligence products for different needs. The products are used to make graphs, tables, and other visualizations from various data sources. The products are advanced to use and the flagship product even has a proprietary domain-specific programming language for data manipulation. Since the company is more than 20 years old, the products are in different stages of the life cycle, albeit all of them are still offered. The company is a large enterprise with about 3 000 employees and 500 of them in the development department. The company has grown rapidly during the latest years with an increase in the number of employees. The sales department is very large and uses primarily direct sales to other businesses. The company has a team specialized in user experience research that gathers qualitative feedback on a regular basis. The company conducts qualitative experiments to evaluate both prototypes and completed functionality with users. However, not all development teams are on board with this yet so not all features are evaluated in this way. The user experience research team is aware of and interested in quantitative experiments but getting such systems in place has not been a company priority.

3.3.12 Case L: Employee management product

The final case company develops a product for employee management. The product is intended to be embedded in the customers' intranet and has both administrative and end-users within each customer organization. The company was founded 20 years ago and has about 250 employees of which about a third in the IT department. The company relies on direct sales with a sales force and most of the revenue comes from projects that integrate

their product at customer sites. The company pays close attention to user experience and has a team involved with user research that conducts qualitative experiments with prototypes. Since the product is still considered to be early in development this experimentation is somewhat frequent.

4 Theory Formulation of FACE

FACE describes factors that affect continuous experimentation and how these factors contribute to an organization’s ability to gain value through continuous experimentation. An overview of FACE is provided in Figure 2. In essence, the theory states that to achieve gains from continuous experimentation, there should be sufficient processes and tools for CE to conduct experiments (P1), the problem the software solves needs to be sufficiently simple to be measurable (P4), simplifying said problem complexity requires business model changes (P5), and the business model should be such that it provides incentives to conduct experimentation (P6). The gains are achieved through conducting experiments that improve the problem-solution fit (P2) and product-market fit (P3). In this section, the theory is defined by describing its constructs, propositions, scope, and its validity is discussed. Empirically-based explanations and expanded explanations of the theory are provided in the succeeding Section 5 from our 12 case companies.

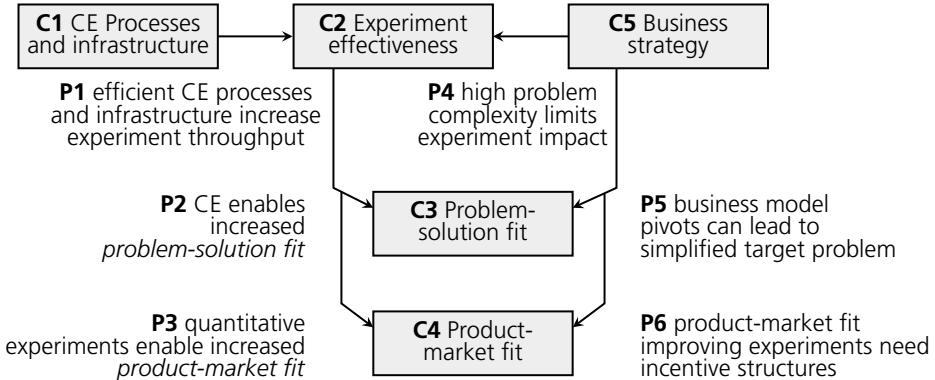


Figure 2: The constructs and propositions of the Factors Affecting Continuous Experimentation (FACE) theory. Experiments (C2) can make improvements in problem-solution fit (C3) and/or product-market fit (C4). CE processes and infrastructure (C1) increases the amount of experiments that can be done. Business strategy (C5) is crucial for having incentives to improve the product.

4.1 Definitions

The following terms are used to describe the constructs and propositions of FACE:

Experiment is an activity that introduces a *change* in an *offering* with the *goal* of learning or improving the offering based on feedback with *user data*. The experiment should ideally be carried out in a production environment with real users to get high fidelity results. The term covers various types of experiments such as quantitative controlled experiments (A/B tests) and quasi-experiments, and includes less rigorous prototype experiments where the users' ability to use a specific feature or willingness to pay for a feature is evaluated qualitatively. There must always be a way to decide if the *goal* of the experiment is met or not. In the case of a controlled experiment this goal is quantitative and assessed using metrics derived from *user data*.

Quantitative and qualitative experiments are different types of experiments. Qualitative pre-deployment experiments are done by implementing a fast prototype and evaluating it on users. The fidelity of the prototype can be anything from a hand-drawn sketch to almost completed functionality. It is hard to get sufficient numbers of users to evaluate a prototype quantitatively in a controlled experiments, so pre-deployment experiments are usually evaluated with qualitative data. Quantitative post-deployment experiments are executed live in a production environment with real users. As such, post-deployment experiments usually have higher infrastructure needs.

Continuous experimentation is the holistic process, encompassing the whole software engineering process and involving multiple experiments conducted in iterative cycles. Based on observations from the interview material, we deduce that experiments are described as *continuous* for two reasons. First, an initial *experiment* is not always decisive, and many experiments in a row might be needed to refine a *change*. Second, the experiment result might uncover new knowledge about the offering, users, or business that begets further related inquires or even new features.

Experimenter is the person initiating and or being in charge of the experiment. It is not a role per se, as there are multiple roles involved in various stages of an experiment (see Section 2.2).

Change refers to a modification to an *offering*. In traditional statistics or medicine literature this is called a *treatment*. It can be a new functionality in the software (i.e. a *feature*) or a modification to an existing one. The change can be in any part of the software, it does not have to involve changes visible to users such as on a user interface. The scope of the changes we have observed in the interview material varies, from a small tweak of the font-size on a button to rebuilding a large part of the product. For some companies it corresponds to a commit to a version control system [164]. Though, it is recommended that the change should be as small as possible while still potentially achieving the goal in order to minimize risks [106, 172].

Goal is what the experimenter wants to achieve with the experiment. In a quantitative experiment, the goal can be described with an improvement in a metric. The term *goal* is preferred in this work, since the more specific term *hypothesis* implies a deeper thought behind an experiment, which is not always the case. For example, when Google tested 41 shades of blue for their links [141], the goal was to find the colour that got the most clicks, but the learning obtained is limited.

Offering is a catch-all phrase referring to what software a company is developing, such as a product, service, web shop, etc. This term is used when the distinction between these are not important.

User data refers to data obtained from actual users in a production environment. User data could also be obtained through other means, such as questionnaires, interviews, or by eye-tracking. There are three categories of user data: (1) *user experience data*, such as clicks or session durations; (2) *sales figures* from the sales process, such as whether users purchase a license to the product; and (3) *computing performance data*, such as CPU usage.

User experience metrics are derived from *user experience data* that measure the user experience, such as degree of users that engage with the software under experimentation, time spent on parts of the software, rate of users that complete a certain goal etc. All software with users have potential access to this data.

Sales metrics are derived from user data originating from the sales process of the offering. The sales metrics include revenue from sales or subscriptions, churn of subscribed users, conversion rate of users to paying users, etc. We draw a distinction between *sales figures* and other *user data* due to their direct impact on business. However, not all companies can use sales metrics in experiments.

Proxy metric is a metric that substitutes for another more relevant metric that cannot be used directly in an experiment for some reason. For example, consider an e-commerce web shop with insufficient traffic to obtain statistical significance on conversion rate or other sales figures. They could use clicks as a proxy metric for purchases since far more users click on products than buy them. However, the signal-to-noise ratio would be much lower and an increase in clicks could even lead to a decrease in conversion rate if the user experience becomes more complicated as a result of the change (requiring more clicks). As such, the assumptions made when choosing a proxy metric should be continuously verified.

Business model is the way a company has structured their activities, offering, revenue streams, costs, etc., to obtain and give value to users inline with a business strategy, see Section 2.1.

Pivot is a major change in the business model done in order to better realize the business strategy [53, 168]. The distinction we draw between *changes* and *pivots* is that pivots have a larger scope and are not based on evidence to the same degree as a change that is part of an experiment. A pivot can be done for various reasons, such as to find a different user base or to overhaul the offering in a major way.

4.2 Constructs

FACE identifies that both CE processes and infrastructure (C1) and the complexity of the problem that the offering solves for users (C3) have an effect on an organization's ability to experiment effectively (C2), in a way that affects product-market fit (C4). Furthermore, FACE states that an organization's business strategy (C5) plays an important role in facilitating experimentation directly through incentives and indirectly through simplifying the offering.

C1 *CE processes and infrastructure* are software engineering practices enabling software experimentation primarily through an experimentation platform and a process for conducting experiments. Since experimentation involves implementing software changes, this construct also encompasses standard practices for efficient software engineering, such as software testing and quality assurance, while it is not the focus of this study.

The technical infrastructure includes a continuous integration and continuous deployment (CI/CD) pipeline that enables experiments to be executed without much delay, infrastructure to store and extract data, and an experiment platform to manage and orchestrate experiments running in parallel. The organizational infrastructure involves having access to all required experimentation roles. The processes entail designing and conducting experiments, and also setting experiments in the overall context of a company. For example, how hand-over is handled between experiments initiated by data scientists or marketers to the software development department. Conducting experiments requires some rigor in the processes, such that experiment results are accurate and trustworthy. The processes do not have to be explicitly documented to be rigorous, as long as they are followed.

C2 *Experiment effectiveness* is the degree of effective experimentation conducted in an organization. It is a function of the throughput of experiments, the impact the experiments have on users or business, and the ability to measure it accurately. Throughput is also further divided into speed and capacity, where speed is the time it takes for an experiment from design to decision and capacity is how many experiments can be handled simultaneously. Throughput is a function of both the organizations' capabilities and user data availability.

- C3 *Problem-solution fit* is the degree to which the offering solves users' problems. Problem and solution is part of the lean canvas business model [201, 240]. There can be multiple problem-solution pairs in the same offering and the solution can address multiple problems. Whether the problem-solution fit can be measured depends on the specific offering but is usually measured (or at least measured with *proxy metrics*) through *user experience metrics*. The problem-solution concept is identical to the one in design science [306, 255], which is a research paradigm for prescriptive research such as tool designs.
- C4 *Product-market fit* is “[a] measure of how well a product satisfies the market.” [219, p. 7]. This construct describes the offerings ability to generate economic value (usually revenue from sales), i.e., that there is a market for it. Product-market fit is advocated in lean startup as the ultimate target to strive to improve in for all companies [201], not just startups. It might not be easy to measure product-market fit directly in an experiment, but it can be obtained through customer surveys [240]. Sales figures in the form of user retention, user growth, etc., serve as *proxy metrics* for product-market fit. As such, growth is included in the construct.
- C5 *Business strategy* is defined as follows: “*strategy is the direction and scope of an organization over the long-term.*” [154]. A *business model* is a concrete plan for how to realize the goal of the business strategy. As such, the choices made in the business strategy will have effects on the whole company. This includes, for example, the licensing and revenue model under which the software is sold, how the software development organization is structured, and what channels are used to reach and acquire customers.

4.3 Propositions

The propositions describe how an organization's circumstances affect their ability to perform continuous experimentation (P1 and P2) and in turn indirectly (P3) or directly (P4) affect their product-market fit through continuous experimentation (P5).

- P1: C1→C2 *Efficient CE processes and infrastructure increase experiment throughput.* The existence of processes and infrastructure (C1) that support CE increases an organization's experiment effectiveness (C2). For example, a continuous experimentation platform could enable performing parallel experiments, thus increasing the number of experiments, or an efficient continuous integration and continuous deployment pipeline could ensure that deployments happen without delay, thus increasing the speed of experiments. Additionally, the degree to which an organization has prepared their data infrastructure for collecting and analyzing data in their software offering and sales process determine the extent to which they can experiment on those parts.

As for processes, correct prioritization of experimentation ensures that the changes with the highest potential impact on problem-solution fit (C3) or product-market fit (C4) are selected, that the experimenter can trust the results, that the metrics are relevant, etc. Experimentation is an activity that encompasses most aspects of software engineering, from prioritization of features to post-deployment, and any delay to any of those activities will cause delays in experimentation too. Hence, there are many opportunities for improving the effective experimentation throughput (C2) with processes and tools or other infrastructure.

P2: C2→C3 *CE enables increased problem-solution fit.* Experiments (C2) that target problem-solution (C3) improves the user experience, i.e. the ability of users to solve problems with the product. Either pre-deployment prototyping experiments and post-deployment controlled experiments can be used for this purpose.

The process of conducting experiments (C2) over time to optimize the product with a defined goal will lead to improvements in the product based on that goal. This is the main benefit of CE. The impact of each individual change might not be great. However, since changes in an experiment are only finalized if they have a positive impact, then the accumulated experiments conducted over time will have a positive effect. There are diminishing business value returns to how well the problem-solution can be fit. Even if the solution perfectly solves the users' problem, it does not mean that the problem is important and that users are willing to pay for it.

P3: C2→P4 *Quantitative experiments enable increased product-market fit.* The product-market fit (C4) measures how well the product can meet market needs and generate revenue, hence it is the ultimate goal of for-profit products or services. The companies that are able to affect product-market fit can use CE to great advantage. However, product-market fit is more difficult to target than problem-solution fit due to having to affect customers' purchasing intent. The effect of software changes on product-market fit is usually subtle and so requires precise measurements that only quantitative data can serve, e.g., with controlled experiments (C2). Not all companies are able to affect product-market fit with experiments (see P4 and P6).

P4: C5→C2. *High problem complexity limits experiment impact.* The change being experimented with should have a direct link to the actual user or business value, enabling assessment of the impact of the experiment (C2). Establishing that link can be harder for products solving complex problems (C5) because the product can be hard to modify or the impact of the change cannot be accurately measured. Thus, in order to perform experiments under those circumstances, the possible changes are limited or the impact of the changes will be uncertain, respectively. Post-deployment experiments are more challenging and are thus affected by high problem complexity to a higher degree.

Reasons for the product being hard to modify could be due to not wanting to disrupt users because they need stability, rigid customer requirements and contracts, complex deployment and delivery with on-premise installations to other organizations, or lack of access to software that the product is integrated with. The product can be hard to measure if the usage is hard to quantify, the product is configurable with variants that the proposed change affects differently, or there is no meaningful metric to use. These examples are all aspects of problem complexity of the problem-solution pair from the business model, either directly or indirectly through the ways that the product is built to deal with problem complexity (i.e. configurability).

P5: C5→C3. *Business model pivots can lead to simplified target problem.* CE is primarily suitable to making changes to address the solution part of the problem-solution pair (C3). However, making changes to address the problem is a big endeavour because it changes the business model (C5) and will likely require involvement of the whole company (i.e. software engineering, sales & marketing, and management). There are practical limits to experimentation, such as, when the proposed change is too costly to reverse in case of failure. In addition, according to P4, the presence of high problem complexity itself limits experimentation. Consequently, experimentation might not be useful for making changes to address the problem complexity. Instead, a *pivot*, i.e., a change of direction in the business model that affects the value proposition and thus the product can be required to realize changes to problem complexity. An example of a pivot would be to shift from targeting a broad market to addressing a narrower customer segment consisting of a niche market with a more specific value proposition. This would hopefully reduce the set of required features and thereby reduce the problem complexity and the user experience could be simpler.

P6: C5→C4 *Product-market fit improving experiments need incentive structures.* Problem-solution experiments can be done by finding user problems and designing solutions for them. Software engineers used to an agile user-centered process will be familiar with this [39]. Having experiments target product-market fit (C4) is more challenging, since market considerations are outside traditional software engineering responsibilities. Also, sales metrics might not be available for experimentation depending on the licensing and revenue model. As such, targeting product-market fit requires incentive structures to be in place with a link between business value, user value, and software engineering activities. This can, for example, take the form of an explicit team goal on a target metric that can affect product-market fit and be measured in experiments.

Note that a focus on product-market fit does not come with a degradation in user experience or the users' obtained value of the software. Indeed, if there is a direct connection between the users' satisfaction with the software and their inclination to pay for it, then sales figures can be used to simultaneously optimize user *and* business

value. This is the case for software provided as freemium (where the user can use a free or a paid version of the software), as a demo/trial versions, as a subscription, with in-app purchases, etc. This is not the case for business models where the user pays once upfront for the product without first using it.

The following consequences on product-market fit experiments happen when the incentive structures are in place:

- The software engineering and sales & marketing departments will have the same incentives and can thus work more easily together. This is also evident from the rise of growth marketing/engineering in industry [162, 288].
- The prioritization process can use metrics from the sales process to inform development and can chose to develop features they believe will best benefit users and business.
- Focusing on acquiring new users will lead to having more user data which will increase the ability to do experiments further.

4.4 Scope

The scope within which FACE is applicable is user-intensive software companies with a user-facing offering. For example, media content services (Case D), e-commerce web shops (cases E, H, and I), and application software products (Case K). Not all companies in the study have software development as their primary activity, but software is a central part of their business model. The theory is derived from empirical observations on those companies and it is unknown whether the theory is applicable or useful outside this context of software intensive companies, for example, non-profit organizations developing open-source software, companies developing embedded systems, companies that are involved with experimentation from a marketing perspective but that do very little software development, such as a news website or web shops without IT departments.

4.5 Theory validity

We evaluate FACE using the criteria proposed by Sjøberg et al. [274], namely testability, empirical support, explanatory power, parsimony, generality, and utility.

Testability. FACE makes high level claims about company processes, organization, etc. As such, the ability to use experiments to test the theory is low due to the large scope that would be required of such an experiment. However, the theory makes claims about real world phenomena and those claims can be validated by using it to analyze and explain CE practice at other companies.

Empirical support. The theory is based on an extensive multi-case study with 12 cases of various contexts and 27 interviewees with various backgrounds and roles. The cross-section of empirical underpinning per case and proposition is shown in Table 3. Each proposition is supported by evidence from multiple cases.

Explanatory power. While there is no ability to make quantitative predictions from FACE, the theory can be used to differentiate CE in real companies and can explain why some companies derive more value from experimentation than others. The context of all case companies that the theory is based on is also available (see Section 3.3) and can be used by practitioners to compare with their own organization's context, and thus judge the applicability of the theory for that context using theoretical generalisation. The theory also uses terms and concepts from established pre-validated knowledge, such as lean canvas from lean-startup [201, 240].



Parsimony. The number of constructs and propositions have been continuously reduced and combined during the theory building process. The remaining constituents of FACE are needed to explain the data from the cases.


















































Generality. The scope of the theory is user-intensive companies which limits the generality of the theory to such companies. The case companies on which the theory is based are quite different in terms of size, age, and domains (though e-commerce is over-represented) so the breadth of the scope is wide.

Utility. FACE can be used by software organizations to understand their ability to perform CE and the factors that influence this. The interviewees in the study were generally very interested in learning more about CE which hints at the overall industry relevance, in addition to the large number of industry authors active in the research about CE [9, 247]. The guidelines that are derived from the theory show what state-of-the-art CE is and how companies can elevate their experimentation based on their context.

5 Theory Explanations and Empirical Underpinning

In this section, the supporting evidence from each of the 12 cases, for each of the six proposition in FACE is presented in order, along with an expanded explanation of the meaning and impact of each proposition. We refer to theory constructs as C1–C5, and propositions as P1–P6. Note that, only the salient evidence is discussed in the text. See Table 3 for a complete picture of the supporting evidence per case and proposition. The strength of the evidence is judged qualitatively based on how much and how direct the propositions are discussed at the interviews. As such, it does not reflect, e.g., how good or bad the cases are at conducting experiments (P1).

Table 3: Cross-section of empirical underpinning per case and proposition in FACE. Cells are marked  for proposition-case pairs with strong evidence and  for pairs with only some evidence.

Case	Proposition					
	P1	P2	P3	P4	P5	P6
A E-commerce algorithms						
B Local search						
C E-commerce consultants						
D Video Streaming						
E Web shop						
F Customer relations						
G Engineering tools						
H Web shop						
I Web shop						
J Product information						
K Business intelligence						
L Employee management						

5.1 CE processes and infrastructure at the case companies (P1)

Companies in the study conduct and depend on CE to different degrees (see Table 1). The case companies also have matching degrees of processes and infrastructure support to match their level of experimentation (with some exceptions discussed in this section). All interviewees at companies with frequent experimentation mentioned efficient process and infrastructure (C1) as crucial for efficient experimentation (C2). The analysis on P1 is divided into three parts: (1) impact of efficient processes, (2) impact of efficient infrastructure, and (3) the impact of low throughput.

The four companies with High experimentation expertise in Table 1, cases A, D, G, and I, also have *efficient processes and infrastructure* to support their experimentation. These cases have a data infrastructure, experimentation platforms built in-house, all relevant roles, and have established processes for CE. *Case A* has low demand on experiment throughput and instead focus their efforts on advanced statistical techniques and speed of experimentation. *Cases D and G* do experiments on all their parts of their software products and have both advanced techniques and a streamlined processes for CE. *Case I* has similar advanced experimentation at some teams in the company. As expressed by interviewee I4 who’s team conducts large amounts of experimentation: “If you have a team that has [experimentation] in its soul, then you want a streamlined process for how to conduct experiments.” Developers

in case I should be able to put an idea under test within hours of its inception and have experiment results a few days later. They struggled initially with their commercial-off-the-shelf experimentation platform because the data volume overwhelmed it, until they built their own.

Two case companies in the study in particular *struggled with implementing the processes and infrastructure efficiency* they desired, cases E and I. *Case E* has an ad-hoc software engineering process which causes issues for their experimentation. According to interviewee E1, there is a constant change of direction from management in prioritization and what metrics to optimize for, incorrect experimentation execution such as stopping experiments too soon or not using statistical tests, and management not taking account of experimentation results. This made many experiments ineffective (C2). Some of these issues could also be attributed to organizational culture, but the way the issues manifest is through lack of adhering to process. *Case I* had two of the teams (responsible for search engine and recommendation engine, respectively) working efficiently by having their own purpose built experimentation platform for their part of the product, as described above. The other software engineering teams at case I, about 50, used a central support team with a data science focus that conducted the experimentation independently. This was described as “*totally unreasonable*” by I3 due to the low experimentation throughput, caused by both technical limitations in the commercial cloud-based experimentation platform and the overhead of having to involve another team.

The amount and quality of available user data was a frequent topic of discussion that was discussed to some degree at all cases. User data is not an explicit part of FACE, but there are processes for adapting experimentation to low availability of users data. Both the amount and quality of user data has a direct impact on the experimentation throughput (C2). A certain pre-determined number of data points needs to be collected to get some degree of certainty in the results. Low quality on user data lowers the information that can be learned from each data point, thereby also lowering experimentation throughput. User data is a limitation at companies with vast amounts of user data too, since some experiments only target certain users (e.g., that has a certain feature enabled or belongs to a certain user segment).

5.1.1 Impact of efficient processes

Experimentation requires following a rigid processes to ensure trustworthy results. This is important for both qualitative pre-deployment experiments and quantitative post-deployment experiments. In qualitative experiments there is much manual and subjective work that must be conducted consistently across users and experimenters. For quantitative experiments the tools must be used correctly to get accurate numbers. In addition, there are many methods and techniques that can be used to get more efficient experimentation:

- Having *data-driven prioritization* ensures that the most important experiments are done first (done at cases A, D, G, I, and J). This could be done by analysing historic user data to see where users have issues, by surveying users, or by properly splitting developments into the smallest possible implementation (such that it can be aborted early in case of failure).
- Using *data mining* after experiment results can be used to get more information out of experiment results, such as dividing the users into segments and analyzing whether the results differ in the segments (done at cases A and G).
- Making *power calculations* to figure out how many data points are needed in an experiment is a recommended step to do before the experiment is started [172]. However, all interviewees except the ones at case G admitted to never doing it or using the same calculation for all experiments.
- Using both *qualitative and quantitative experiments* is recommended. Many companies only have infrastructure for one type of experiment, but cases D and I have both and are able to use the experiments that best suit the situation at hand.
- Optimizing the *statistical test* to a more precise version that suits the given situation increases chances of obtaining statistically significant results (done at cases G and I). However, G3 also warned about spending too much time on this since it requires specialized knowledge and is technically challenging.
- Using *experiment designs* with multiple variables achieves more nuanced results, such as multi-variate tests or multi-armed bandits (done at cases A and G). This also requires specialized knowledge and is technically challenging.

5.1.2 Impact of efficient infrastructure

Getting started with CE was not described as technically hard by any of the interviewees. As phrased by interviewee G3, “*The original experimentation system was like 15 lines of Scala, it was just really really simple. It’s interesting how you can start up really easy.*” Also, the requirements for pre-deployment experiments is even lower because the qualitative methods do not rely on technical infrastructure. However, as post-deployment experimentation is scaled up the demands on infrastructure (C1) increase to keep up with having efficient experimentation (C2).

When the case companies discovered the value of CE they steadily increased their frequency of experimentation to cover more of their new developments and also on old features that has never been subjected to experiments. The infrastructure is a bottleneck to enable increased experimentation but all cases except H were able to keep up with increased infrastructure demands due to CE being prioritized in the organizations. Case H is lacking in resources to increase their infrastructure.

The overall *infrastructure* was similar at all cases albeit at different levels of maturity. The infrastructure includes three parts. (1) *Data infrastructure* to store and provide query support for user data. This includes telemetry of user data and product data in all parts of the software, collecting various information about users for segmentation, and a system for storing and accessing this data (i.e., a data warehouse). In addition, all data needs to be stored securely and in compliance with legislation (i.e., GDPR). (2) An *experimentation platform* with support for starting and stopping experiments, configuring which metrics to target and what additional metrics will be monitored, support for segmentation and arranging metrics in hierarchies if there are too many, alerting in case things go wrong, ways of running experiments in parallel (non-overlapping in case their changes are conflicting), etc. Finally, (3) *competences* to develop and support infrastructure and to conduct experiments (see Section 2.2.1). Also, the developers need to be educated in CE if they are to take part in it, which was a challenge for the cases it happened at (D, G, and I).

Of the three infrastructure parts, *data infrastructure* is the most demanding to develop, according to interviewees at D, G, and I, because data infrastructure must be implemented in the entire software offering, rather than as a standalone development. Investment in data infrastructure is one of the reasons that *case A* is able to conduct experimentation with High expertise despite their small company and software department size. Their product relies on user data for algorithms, such as recommender systems that needs the same infrastructure, and they were able to use it for conducting advanced experiments (multi-variate tests and multi-armed bandit variants, see [249]) even though experimentation is not widespread at the company.

5.1.3 Impact of low throughput on experimentation efficiency

There are additional consequences of low throughput to the impact of experiments (C2), caused by either low capacity or low speed as follows, besides that companies fail to get as much experimentation done as they would like.

A *low capacity* to run experiments increases the risk of releasing features with user-related issues, since only a subset of the desired experiments can be conducted. It is not always obvious what changes will have an impact on users or not. This was something all case companies experienced except the most advanced ones (cases G and I). As phrased by AI:

“A/B tests are always unpredictable, that has been proven. Again and again by us. We do release some non A/B tested functionality and I can’t be totally sure it is all good, but it is what it is. [...] We always focus on what customers need as much as possible and do our A/B tests on that.” Low capacity is primarily caused by lack of user data or developer resources.

Low speed of experimentation can be frustrating, as E1 put it when queried about challenges of CE: *“long lead times makes the whole being data-driven thing almost impossible”*. We identified the following two additional consequences of low speed of experimentation. (1) Low experiment speed can result in other changes in the product or market to invalidate the result. Due to, for example, changes in company priorities (mentioned by E1 and J1), the experimenter forgetting details about the change (mentioned by A3 and D1), or conflicting software changes that make the change incompatible (mentioned by A3, A4, and K2). In addition, (2) it will be hard to use the insights to form hypothesis in the follow-up projects if they are started before the previous projects’ experiment is completed. Thus, experimentation is not really *continuous* when experiments are slow (mentioned by B1).

The issues with lack of speed were primarily observed at *Case A* due to challenges with continuous delivery, at *Case B* due to having experiments with very large scope, at *cases D and J* due to their recent start with CE thereby having lots of various inefficiencies in their process, at *cases C and H* due to low user data volumes, and at *cases K and L* due to using primarily qualitative methods with much manual work.

5.2 Experimentation impact at the case companies (P2 and P3)

All cases have experiences with CE for increasing problem-solution fit (P2). However, not all companies are able to affect product-market fit. Only cases D, G, and I experiment with product-market fit regularly. Cases B, E, and J target product-market fit only to some extent, cases B and J have multiple user groups of which they are only able to target one with experiments, and case E have issues with CE processes. The remaining cases do not target product-market fit due to propositions P4 and P6.

Sales metrics (revenue, conversion rates, churn rate, etc.) were mentioned often in the interviews as an appealing metric to use for experimentation. As explained by D1, it is both directly relevant to business, and users presumably only pay for software that they think fulfills a need for them. As such, when an experiment can use sales metrics, it can optimize the product for both business needs and user needs simultaneously, thereby increasing experiment impact (C2) on product-market fit (C4).

According to P3, only quantitative experiments can target product-market fit, while both pre-deployment prototyping experiments and post-deployment can target problem-solution fit. None of the cases were able to target product-market fit with anything other than controlled experiments with sales metrics. Presumably due to that it is harder to measure product-market fit in a way that cannot be done with only a prototype since it requires users to actually pay for something, hence sales metrics.

5.3 Problem complexity at the case companies (P4)

When the problem (C5) that the product or service solves is complex, it is challenging to experiment efficiently on it (C2). The problem complexity of the offerings at the case companies span a wide range, from low complexity (cases D and G), medium (cases A and K), to high (C and L). By problem complexity, we mean that there are various challenges in the way the offering delivers value to users. We identified three issues with how problem complexity affects experimentation: (1) problem complexity makes changes hard, (2) complex user experience makes measurements hard, and (3) configurability addresses problem complexity but splits experimentation effort.

Some degree of complexity in the user problem can be overcome by using qualitative methods, e.g., user interviews, observations, focus groups, etc. User observations are regularly used at cases K and L and can be used even when the user experience cannot be quantified into meaningful sessions nor specified as an adequate metric. The experimentation process with user observations at case K and L is slightly different than for a quantitative controlled experiment. There is usually no control group, instead the experimenter selects a small group of users and observe their interaction with the change in the product and compares with earlier results (as in a natural or quasi-experiments). As such, qualitative methods can evaluate a change and is considered to be a valid experiment. However, the efficiency of qualitative experiments in terms of reliability per work hour is low due to the cost of interviewing, recording, coding, etc., compared to the cost and scalability of a quantitative experiment once the infrastructure is in place. The ability to be precise in exactly which change has what effect is also lower compared to quantitative experiments. Qualitative methods do have advantages with richer data that can be used to explain why changes fail or succeed. But as a method for conducting an experiment with realistic circumstances they are limited.

5.3.1 Impact of problem complexity on making changes

Some software offerings can be hard to implement changes on, due to having to be integrated with other software, causing communication barriers with their developers, or that there are requirements or expectations from customers that the software should not change. In such cases, the experimentation process becomes hard due to the complexity of implementing the change in the experiment. This affects the ability to conduct controlled experiments (C2).

When experimenting with software with *integration* needs, the experimenter has to interface with software that is external to the organization. The experimenter cannot make changes incompatible to it, or they have to communicate with the software engineering organization responsible for the integrated system, which would cause delays in experimentation. At *Case C* their business is centered around integrating different systems to build web shops, C2 mentions this regarding their customers ability to A/B test on their site: “*What customers can change is actually only content [text and images]. [...] Supposedly, at best, the customers can A/B what it looks like [user interface].*” *Case H* is one of their customers that did do their own A/B testing with involvement from interviewee C1. At *Case A*, where their product is used in other companies’ software, the interviewees complain that they cannot change how their product is being used (sometimes incorrectly) and that deploying the changes takes considerable time for some customers.

Finally, *user or customer expectations or requirements* can hinder CE because some users might not want changes. At most of the case companies, the users are able to overcome changes easily, so this aspect of problem complexity was only discussed by interviewees at case D, K, and L. At *Case D* they mention how their B2B product has customer requirements that mandate the behavior of certain features. It is not possible for them to make changes to these features even if it would be beneficial to their other customers. Of the other B2B companies in the study, only *cases D and F* accept customer requirements to their product in this way. At *cases K and L*, the interviewees mentioned that their users do not want frequent changes due to their software being used in a corporate setting where users do not want changes to disrupt their work flow.

5.3.2 Impact of product complexity on defining measurements for controlled experiments

The user experience of software with a complex and open-ended work flow cannot be neatly quantified into chunks or summarized with a single metric. Sales metrics can sidestep the issue of defining a user experience metric, but not all companies can use it for experiments directly and it might not be a suitable target for all experiments. Cases F, K, and L cannot use sales metrics at all due to their business model (see P6) and they also have complex

and open-ended user experiences. All of the other case companies mention using metrics that are related to the user experience as a target metric to improve problem-solution fit. User experience metrics are also monitored on experiments that target product-market fit to ensure that business value does not come at the expense of users at all cases able to target product-market fit.

The cases *F*, *K*, and *L* have similar underlying reasons for having a *complex and open-ended user experience* as explained in the interviews. The software is used throughout the day, such that it cannot be neatly quantified. The goal that users have when they use the software is hard for the software engineers to extract and measure, and the number of features are large such that there might not be enough user data on some features to measure it.

While it is always possible to find *something* to measure in the user experience of all software, such as number of clicks or other user interactions, it is far from certain that optimizing those metrics will transfer to concrete gains. According to L2, good user experience metrics should measure whether the user is able to accomplish their goals with using the software or not. Clicks was specifically mentioned as often being a “*terrible metric*” by A4 and G3 due to being able to be too easily influenced by experiments without having a real impact on more relevant measures. Intuitively, the number of clicks should be kept low to have an efficient user experience. But if the program is something users enjoy spending time on (and revenue is earned through that), then the number of clicks will be beneficial to increase instead.

5.3.3 Impact of configurability on experimentation

When software is built to be configured into multiple unique variants (as in software product line development [233]) the problem of having enough user data for each feature is exasperated. In such cases, the amount of users for each such unique configuration is lower than for the total set of users, which leads to experiments taking longer to complete due to having less amounts of user data per time period. Having software be capable of configuration is viewed here as a consequence of dealing with high problem complexity. *Case A* has a product with some degree of configuration. They do experiments at different customers and analyze them independently. However, they have a limited number of customers so it is not described by the interviewees as very challenging for them to handle. *Case F* has a product with a very high degree of configuration and they are not able to conduct experimentation partially because of that. They have a lot of features in their product and the features exist in multiple variants and experiments would have to be repeated for the different variants, which is not efficient or might not be feasible.

5.4 Business model pivots at the case companies (P5)

According to P4, there are limits to what can be done with CE when the problem complexity is high. Companies can simplify what problem the software solves by pivoting their business model (C5) to solve another problem that can lead to a higher problem-solution fit (C3). CE is not suitable for all development tasks [36, 205] and is unlikely to help reduce problem complexity significantly; we see two reasons for that. First, while CE can improve problem-solution fit, actually changing what *problem* the software solves is a large change that goes beyond the scope of experimentation. Changes in an experiment need to be sufficiently small such that the change can be reversed in case of bad results. Second, high complexity reduces the ability to experiment (P4), which is a catch-22 scenario; experimentation requires a goal to target but no goal can be specified due to the high complexity.

Case company D pivoted recently by changing their target users from business customers to private individuals. After the pivot their prioritization was data-driven to meet market needs and they had less customer requirements on their user experience, thus they were able to simplify the user experience. They also increased their experimentation significantly as a result of the pivot. Note that, the pivot that case D conducted was not done specifically to enable experimentation. Rather, the goal was to reach a larger market which needed a simplified product, and enabling experimentation was a side-effect of simplifying the problem complexity. No other company in the study has gone through a similar pivot, but A5, C2, and F1 mentioned pivoting (though not necessarily with that term) in regards to what changes they would have to implement to enable experimentation in their company.

5.5 Experimentation incentive structures at case companies (P6)

Some companies in the study have business models (C5) that are suitable to experimentation (C2) targeting product-market fit (C4). These business models provide incentive structures that enables experimentation, e.g., by having metrics available that software engineers can use to directly affect the product sales and user growth. Part of the differences in incentive structures is explainable with the *sales-led and product-led growth dichotomy*, see Section 2.1.2. Companies with a business model with product-led growth rely on the product to obtain customers, while the companies with a sales-led growth rely on a sales department to obtain customers. Incentive structures impact experimentation in three ways as explained below: (1) user-business alignment, (2) sales & marketing interplay, and (3) increased user-data.

Not all companies' offering revolve around a product or service, e.g. e-commerce companies, and those companies cannot be placed in either category of growth. From a software engineering perspective, the *web shop cases E, H, and I*, are somewhat in between product-led and sales-led because while they can use sales figures in experiments, changes in the

software have a limited ability to affect the web shop sales. *Cases B and J*, have multiple user groups with different sales and growth strategies for either role, placing them also in between sales-led and product-led. As such, *cases A, C, F, K, and L* have a sales-led growth, *cases B, E, H, I, and J* have neither, and *cases D and G* have product-led growth.

5.5.1 Impact of user-business alignment

According to P3, affecting product-market fit (C4) relies on quantitative experiments (C2) that target sales metrics. While the availability of sales metrics is necessary to experiment on product-market fit, incentive structures are also necessary. Cases A and C are examples that have sales metrics from their B2B customers, but cannot target product-market fit with those figures since an increase in their customers sales figures does not come with direct business value to them. As such, experimentation to improve the sales figures is only done for product-solution fit purposes at *Case A* and not at all at *Case C*.

In contrast, *cases D and G* uses various sales metrics extensively in experiments, presumably due to their user-business alignment. At both cases, the sales metrics come from software sales and they also both have a subscription-based license model where the user pays continuously to use their service. G3 phrased it as such: *“Our incentives are mostly aligned with our customers’ because it wouldn’t mean anything if somebody purchases [our flagship product] and then decide that they hate it and a week later they cancel the subscription.”*. D2 also expressed the importance of user-business alignment: *“So the way you build things for freemium is that it has to solve a user problem, that’s part of the product breed, that’s part of why you do the thing, all your metrics align to that.”* Freemium is a license variant where the product has a free and a paid premium version, see Section 2.1.2.

Cases D and G both also uses metrics from the sales funnel to find and prioritize issues with their product. As explained by D2: *“There’s industry standard metrics, like the AARRR model Acquisition, Activation, Retention, Revenue, and Referral. We use them to kind of analyze our platform and our product and see where we are missing things.”*. The AARRR Pirate Metrics Framework was proposed by McClure [202] and covers multiple business model aspects.

5.5.2 Impact of development and sales & marketing interplay

Companies with product-led growth models will have the same incentives on the software engineering department and the sales & marketing department: to increase the sales figures. This makes it easier for them to cooperate. At *Case D* they had a few individuals with the hybrid role of growth engineer that worked as an intermediary between the two departments. At *Case G* they included growth marketers in their specialized team that conducted experiments. The growth engineers/marketers were primarily tasked with finding and analyzing hypotheses about the product to experiment on.

5.5.3 Impact of increased user-data

The cases with a strategy (C5) of finding many customers and growing rapidly will get access to more data from their increased user base. User-data volume is critical for experimentation throughput to affect problem-solution fit and product-market fit. The interviewees at cases with sales-led growth (A, C, F, K, and L) all mentioned a focus on large important business customers that can pay more instead of many customers, which gives a higher return on the manual work that the sales force put in. Another difference lie in that the product-led growth cases (D and G) both have a freemium license model that further increases user-data, since the users that use the software for free contribute with user-data.

6 Discussion

We have presented a theory of Factors Affecting Continuous Experimentation (FACE) on what contextual factors in a company's business model and software organization affects continuous experimentation (CE) and how. The theory is based on empirical observations in 12 case companies through semi-structured interviews and subsequent theory building. In summary, the theory states that processes and infrastructure increase experiment throughput (P1) and experiments can increase problem-solution fit (P2) and quantitative experiments can increase product-market fit (P3). However, experimentation is limited by how complex of a problem the software solves for users (P4), the problem complexity can be reduced by pivots in the business model (P5). Finally, experimentation on product-market fit requires business models with the right incentive structures that connect business and user value (P6).

There are two discussion points: (1) what the limitations of CE as a method for software engineering are and (2) which factors affect a company's ability to conduct CE.

6.1 What are the limitations of CE?

There are *limitations to what can be achieved with CE*. According to best practice in continuous experimentation (see Section 2.2) each experiment should be split into the smallest constituent to avoid having to do unnecessary implementation work in case the change is bad. As such, continuous experimentation entails incremental work. In the road-map for continuous software engineering, Fitzgerald and Stol [113] argue that sometimes abrupt changes are needed when creativity and innovation is involved, because incremental work constraints the creativity to similar solutions to what currently exists.

Furthermore, in CE, experiments are ultimately used as a method for optimizing software towards a given goal by taking small steps in the right direction. However, there is a risk of getting stuck in a local optima where no individual small change can improve the current software design, but there might be a better solution to be found if a larger change is introduced. Larger changes are pivots, the impact of which in FACE are described by the proposition P5. When a pivot is introduced to a product that has been polished with experimentation, it is quite likely that the new version of the product performs worse because it is less polished. There could be minor problems that over time could be improved with experimentation to find a better solution. Whether or not it is worth the effort of maintaining multiple versions of the software product during this period is ultimately a business decision that cannot be settled within the confinements of CE.

The value provided by experimentation diminishes over time. When experimentation is first introduced at a company there are low-hanging fruits to experiment on. At the case companies, A, D, E, G, I, and J, there were long held assumptions about the product and customers that were finally able to be tested and this led to some surprises. However, over time experimentation becomes established at the company and more parts of the product are stabilized. Also, the size of experiments will become smaller since the experimenters become more adept at reducing the scope of the experiments, which has also been reported previously [174]. Hence the utility of each individual experiment tends to decrease unless a pivot happens.

Finally, FACE cannot make claims about applicability of CE to companies outside the defined scope. However, we argue that an experiment-driven approach to software development is unsuitable for much of the software outside the scope. That is, software that is not user-intensive (e.g., company intranet websites) or not user-facing (e.g., low level software libraries). Other requirements elicitation techniques or software performance optimization methods might be better suited for these examples, such as using profiling tools to pinpoint performance bottlenecks with low lead time.

In summary, CE can increase problem-solution fit and product-market fit. However, CE should not be the only strategy for improving software at companies due to the limitations of incremental work. Pivots also play an important role in enabling the product to be optimized with CE.

6.2 What factors are at play in CE?

In this study, we show that CE is used for different purposes: problem-solution fit or product-market fit. Similar observations has been done before. Schermann et al. [265] and Ros and Bjarnason [247] describe that experiments are either regression-based to verify or validate features or business-based to optimize. Our results highlight that most companies are only able to experiment with improving problem-solution fit and few are able to affect product-market fit. However, according to the entrepreneur Andreessen [6]: “*The only thing that matters is getting to product-market fit*”. Our main research goal in constructing the theory is to find the factors that explain differences in why companies can apply CE to different effect. For example, why some companies are able to experiment with product-market fit and others cannot.

From previous research, we know that the availability of user data is the main limitation [172] and that offering software-as-a-service (SaaS) [176] facilitates easier experimentation. Also, there is substantial work conducted on the challenges of applying experimentation in a B2B setting [242, 247, 317] and in cyber-physical systems [33, 122, 197]. FACE synthesizes these findings; in fact most of these studies are all on aspects of complexity in the problem the product solves for users; corresponding to P4. FACE highlights additional factors as follows.

Three of the propositions (P1, P4, and P6) affect companies’ ability to use CE to increase problem-solution fit or product-market fit and those form the factors. The first (P1), on CE processes and infrastructure, covers many aspects and not all are regarded by us as very impactful on whether a company can apply experimentation or not. While the throughput would be lower without sufficient processes and infrastructure, it is unlikely to be a complete blocker to experimentation. In addition, several of the companies (see Section 5.1) have described how they gradually improved their experimentation support in parallel with ramping up experimentation. Data infrastructure is an exception that requires significant investment. The final three *derived factors* are thus:

1. *Data infrastructure* is needed to be able to use the metrics that are desired and to cover telemetry of all parts of the software. Companies that rely on user data for other parts of their products, such as recommendation systems, will get a head start on experimentation by reusing data infrastructure.

2. *User problem complexity* is the complexity of the problem that the software solves for users. High problem complexity makes quantifying the user experience hard and making changes in the software hard, which severely impacts the ability to experiment. Modifying the user problem is challenging but possible with pivots in the business model.
3. *Incentive structures* are required to provide a measurable link between business value, user value, and software engineering activities such that experiments can affect product-market fit. Companies that do not have access to sales metrics in experiments will derive less benefits from experiments since user experience metrics are hard to define in a way that they can be used for optimization.

7 Guidelines to Practitioners for Conducting CE

Based on FACE, we have derived guidelines for practitioners. The guidelines are given as a list of recommendations on state-of-the-art CE processes and infrastructure and on how to strive towards it. According to Stol and Fitzgerald [278], theories are suited as a starting point to derive practitioner guidelines from.

7.1 State-of-the-art CE practice

Industry leading companies have published information on their state-of-the-art CE processes or infrastructure, such as Microsoft [130], Google [286] and Facebook [107]. Compared to those, two of the cases in our study have achieved equally high levels of experimentation, *cases D and G*. They conduct experiments frequently, on all parts of their software, on most software changes, and on business relevant metrics. CE is an important part of their software development process, they have shifted from a mindset of shipping features to an experiment-driven approach where only functionality that are proven to solve user and or business needs are delivered. They are able to affect both problem-solution fit and product-market fit with their experimentation.

The RIGHT process and infrastructure model by Fagerholm et al. [106] serves as a recommended starting point for practitioners. RIGHT contain no conflicting information to the ones reported here but it does not reflect mature CE. We have observed the following additional aspects of mature *processes and infrastructure*:

- *Prioritization* should be based on user data. Companies use scoring methods to quantitatively prioritize expected impact and difficulty of implementation. Both cases D and G use the Impact, Confidence, Ease (ICE) scoring model [96] and use user data to base the numbers on; when possible.

- *Feasibility analysis* of experiments should be done before experiments are conducted. Only companies with High experimentation expertise (cases A, D, I, and G) conduct this step. It could be as simple as checking how many users use a feature or it could be a power analysis to control false discovery rates [172]. There are similar methods for qualitative research as well [295].
- *Rigorous analysis* should be conducted after quantitative experiments are completed. The cases with High expertise in CE monitor statistical significance on multiple metrics of different types: sales metrics, user experience metrics, and computation performance metrics. The calculations are simple [172], so there is no reason to ignore statistical significance (or effect sizes, or Bayesian alternatives). Proper qualitative analysis is performed at Case K only, with a transcription and coding process.
- *Mixed-methods experiments* are in use at cases D and G, since quantitative and qualitative data complement each other well. Other companies use one or the other. Prototyping pre-deployment experiments can be evaluated qualitatively to pre-validate ideas and to find hypotheses on what to experiment on. Following up with post-deployment experiments give definitive proof. Though, it does require additional roles in the form of both user experience researchers and data scientists to conduct proper analysis of the results.
- *Knowledge sharing* is encouraged at companies with Medium or High CE expertise and is especially important when the experimenters are in a stand-alone team. Knowledge can be shared, for example, in workshops, mails, or internal wiki. One of the strengths of an experiment-driven development method is the opportunity to create domain knowledge through rigorous research. That effort could go to waste if the knowledge obtained from experiments is not curated.
- *Infrastructure* is extensively expanded at the cases with High CE expertise. The *data infrastructure* should support telemetry on all parts of the software and on multiple types of user data and it should be stored in a way that it can be tracked to experiment groups. The *experimentation platform* should support managing parallel experiments, making inquiries into results for different user segments or metrics, and segmentation of users in order to obtain high throughput of experiments. The *competences* should cover the necessary roles of experimentation. In addition, all software engineers need to be trained in experimentation so that they can partake in CE.

7.2 Reaching state-of-the-art CE

The theory identified three factors at play in experimentation, data infrastructure, user problem complexity, and incentive structures. These factors represent hurdles that companies need to overcome to reach experiment-driven development. What follows are the strategies the companies in the study used to achieve efficient and effective CE, also in unfavorable company contexts.

- *Build processes and infrastructure gradually.* While some companies were able to get started and ramp up experimentation (cases A and to some degree B) due to prior investments in data infrastructure, we recommend starting with experimentation practice instead of infrastructure processes. Starting with experimentation is easy, but scaling to high frequency and extent of experimentation is very challenging. The knowledge obtained from continuous experimentation will be useful for continuously scaling up infrastructure too.
- *Seize opportunities to pivot* to simplify what problem the software solves for users. Problem complexity severely limits the ability for experiment-driven development. Case D was able to pivot successfully and was able to change to an experiment-driven process. However, pivoting is likely a challenging endeavor. Failed examples of pivoting might go unnoticed because a likely outcome of a failed pivot is bankruptcy.
- *Use qualitative methods* to experiment in contexts with high problem complexity. Qualitative experiments are much easier to implement since they do not require high numbers of users and hence no deployment and they do not require specifying a quantitative metric. Cases K and L are able to conduct CE by validating software changes using qualitative experiments.
- *Find the right goal* to experiment on that is relevant for both business and users. Metrics are hard to specify since they should stand up to abuse from unintentional attempts to game it, i.e., an improvement in the metric should always result in positive business and user value. For example, clicks are easy to get more of by adding additional steps users must complete, but that results in a worse user experience. Many case companies are only able to use user experience metrics (such as clicks) that can improve problem-solution fit. However, by targeting product-market fit both users' and business' value can be optimized for simultaneously. Companies should put effort into finding the right goal for them, for example by carefully using proxy metrics (see Section 4.1), using qualitative methods, or by providing team goals.
- *Provide incentives with team goal metrics* so that all software engineers are encouraged to start with experimentation and to allow them to take responsibility of the experimentation process. This is easy to implement when working with quantitative experiments since the developers are provided with the necessary tools for achieving

the goals. Cases A, D, I, and G all use team goals. For the companies that do not have the right incentive structures in place they can compromise with process oriented goals, such as goals with how many of new features they are able to evaluate with qualitative experiments, etc.

- *Be mindful of ethics* when providing strong incentives to individuals. Team goals should be set with collaboration between the teams and management to ensure it is good for business and users. For example, one of the experiments mentioned by E1 was to hide the pause subscription button in a menu which drastically improved the churn rate and revenue. This also highlights the need for knowledge sharing since other developers pointed out the questionable experiment.

8 Conclusions

We conducted a multi-case study with 12 companies and built a theory called Factors Affecting Continuous Experimentation (FACE) based on the empirical material for understanding what factors are at play for companies conducting continuous experimentation. Six propositions are included in FACE. (1) *Efficient infrastructure and process improves experimentation effectiveness*. Starting with experiments is easy, but companies can put endless effort into: scaling up experiments, obtaining more insights from experiments, and improving the speed of experiments. (2) *Experiments can affect either the problem-solution fit or (3) product-market fit*. Targeting product-market fit is far more challenging, but preferable, since it can improve business and user value simultaneously. Many companies are restricted to only improve the user experience and thereby problem-solution fit. (4) *The complexity of the problem the software solves for users strongly limits experiment applicability*. High complexity can limit the ability to make desired changes in the software or to quantify the user sessions. Following that, (5) *pivots in the business model is necessary to simplify the problem complexity*, experiments on their own are unlikely to succeed. Finally, (6) *improving product-market fit needs incentive structures* in the form of metrics from the sales process.

FACE can be used to evaluate company contexts to gauge experimentation applicability at their companies. There is still future work to further validate the theory by using it in this way to evaluate companies.

Finally, we present guidelines derived from FACE based on the empirical data from the cases and in relation to related work. The guidelines illustrate state-of-the-art processes and infrastructure and recommendations for practitioners on how to reach towards that when their company has unfavorable factors that affect their ability to experiment.

Acknowledgements

We want to thank all the participating anonymous companies and interviewees for their contribution to this project. This work was partially supported by the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation.

Appendix

A Interview Guide

The following questions should be adapted to suit the interviewees background and role. The nested bullet lists indicate probes that are only asked when the answer to the main question requires clarification.

A.1 Introduction

- Inform about consent
 - Interview will be recorded
 - Free to withdraw at any time (including afterwards)
 - Data will be treated confidentially and anonymized
- Explain purpose of study
 - Investigating context and process of experimentation and data use

A.2 Case context

1. What does your company do?
2. What is the overall business strategy?
3. What is the business model?
 - (a) What user problem does your offering solve?
 - (b) Solution: Product? Service? Consultancy? Other?
 - (c) Who are the customers? B2B? B2C? B2X?
 - (d) What are the key metrics and how is it measured (channels)?
 - (e) Costs and revenue?
4. How stable is the business model?
 - (a) How do you know when to pivot (change direction)?
5. How many employees are there in: Total? Dev? Sales & marketing? Ops?
6. Could you describe your own role(s)?

- (a) What is your background?
- 7. What does your team do within the company?
 - (a) Is it cross-functional?
- 8. In what ways does your company use user data?
 - (a) Is there a specialized data science or engineering team?
 - (b) Could you shortly describe your overall infrastructure for data?
- 9. How does your SE team prioritize what to build?
- 10. How does user data help with prioritization?

A.3 Experimentation process

- 1. How was experimentation introduced at the company?
 - (a) In what department was it started?
 - (b) In what departments is it done now?
- 2. Why do you do experiments?
 - (a) Knowledge? Prioritization? Optimization? Regression? Validation?
- 3. Could you break down the steps that are taken by you and your team when conducting an experiment?
 - (a) Duration? Roles? MVF? Analysis? Power?
 - (b) Are you aware of any missing steps?
- 4. What do you experiment on?
- 5. To what extent do you experiment?
 - (a) Duration? Frequency? Coverage?
- 6. Is there overlapping experimentation?
 - (a) From different teams?
 - (b) Coordinated?

A.4 Experimentation details

1. Could you describe your infrastructure for experimentation?
 - (a) CI/CD pipeline? Reporting? Reliability? Scalability?
 - (b) Do you use blue/green deployment or feature flags?
 - (c) What would you like to improve in your infrastructure?
2. What type of experiment designs do you use?
 - (a) A/A experiments? MVTs? Bandit testing?
 - (b) Do you have decision algorithms taking causal decisions?
3. Do you use any qualitative methods?
 - (a) Focus groups? User studies?
 - (b) Do you have specialized teams or individuals for it?
 - (c) How does it interplay with quantitative experimentation?
4. What types of metrics do you use?
 - (a) What metrics would you want to use?
 - (b) How does your metric translate to company or team success?
5. Are experiments analyzed or executed in different segments?
 - (a) Explicit segments: Verticals? Products? B2B customers? Web pages?
 - (b) Implicit through data mining?
 - (c) How do you handle diverging results?
6. Do you ever do experiments involving external code bases?
 - (a) How did it affect experimentation?
7. How do you share knowledge from experiments?
 - (a) Mail? Meetings? Documentation?
8. Do you do any long term follow ups on experiments?
 - (a) Repetitions? Long-running experiments?

A.5 Holistic experimentation view

1. What are the main challenges with experimentation?
2. What are the main benefits with experimentation?
3. Do you face any ethical dilemmas involving your use of user data or experimentation?
4. How do you strive to improve your experimentation?

A.6 Final remarks

1. Do you have any final comments, anything that should have been asked?
2. Could you recommend us any additional interviewee (or organization)?
3. We will get back to you within 1–2 weeks about a summary of what was said here.

B Code Book

All paragraphs of the text were coded, multiple codes can be used on the same paragraph. Some codes are marked with an X to indicate that they can vary, for example Scenario X should be used as Scenario optimization or Scenario validation, these were expanded freely during coding. The division of detail codes into sections are intended only for improved readability, they are not themes or categories. The first two sections indicate context around experimentation, and the last two are on experimentation process and evidence of actual use.

B.1 Software development and infrastructure

- Testing (*any testing aspects discussed*)
- Prioritization (*any prioritization aspects discussed*)
- System architecture X (*description of type of architecture e.g. embedded or micro services*)
- Data infrastructure (*description of e.g. data warehouses, query engines, data science teams, data engineering teams*)
- Data availability (*whether data is capable to be used for experiments*)

- Data governance (*cleaning, data provenance, data*)
- Experimentation platform (*description of an experimentation platform in use at company*)
- Infrastructure improvements
- Infrastructure challenges
- Organizational structure (*How departments are structured and how much of e.g. sales and development there is at a company with relevance to experimentation*)
- Company culture (*descriptions of culture that influences experimentation*)
- Knowledge sharing (*how knowledge is shared between departments*)
- Technical infrastructure
- Software stack technology
- CI/CD pipeline
- Infrastructure maturity

B.2 Business model and strategy

- Pivoting (*change in business model according to strategy*)
- Product customization (*general or tailored to different market segments*)
- Product complexity (*description of a complex product in e.g. size or user experience*)
- Problem-solution pair (*what problem the product solves for users*)
- Key metrics
- Market constraints (*ethics, legislation*)
- Cost structure
- Revenue stream (*pricing model*)
- Growth model (*how new customers are acquired, channels, customer segments, number users, etc.*)
- Channels (*path to customers*)
- Unique value proposition

- Target customers
- Unfair advantage

B.3 Experimentation process

These codes describe strategies for various stages in the process and why experimentation is conducted.

- Ideation (*how ideas/hypotheses are elicited and prototyping is performed at the company*)
- Experiment design (*how design are decided, pros and cons of designs*)
- Metrics (*which are used, what specific metrics mean, how they are derived*)
- Analysis (*how analysis is conducted*)
- Scenario X (*different experiment archetypes e.g. optimization, validation, verification, learning*)
- Dark patterns (*a dark pattern is an unethical anti-pattern in UX for tricking users*)
- Experimenter X (*used to indicate what role initiates and owns an experiment*)
- Role X (*used to indicate experimentation involvement*)
- Experiment handover (*description of how experimentation is conducted by some specialist and then hand over to product team after completion*)
- Experiment inhibitor (*something hinders experimentation, use in combination with another context giving code*)
- Experiment enabler (*something enables experimentation, use in combination with another context giving code*)
- Experimentation frequency

B.4 Experimentation usage

These codes are used to gauge how much experimentation is done at a particular company, the first three should be mutually exclusive at a company unless there is some conflicting reports by different interviewees.

- Experimentation awareness (*awareness of experimentation at a company but no intent to start*)
- Experimentation intent (*aware of experimentation at a company and want to start experimentation but have not done so yet*)
- Experimentation adoption (*in the process of adopting experimentation and/or increasing scale of exp*)
- Experiment duration
- Experiment goal

These codes are used when a technique is mentioned as being used at the case.

- Sprint experiment (*experiment is conducted as part of ordinary development and is prioritized with other software development*)
- Stand-alone experiment (*experiment that is conducted outside of the development organization*)
- Controlled experiment usage (*A/B test, quasi experiment*)
- Optimization usage (*multi-armed bandits, A/Bn tests, MVT, simulations*)
- Qualitative methods usage (*focus groups, observations*)
- Survey usage (*questionnaires*)
- Data mining usage
- Feature flag experiments (*considerations for running experiments through feature flags in the same deployment environment*)
- Blue-green deployment experiments (*considerations for running experiments through parallel deployments*)
- Repeating experiments (*same hypothesis repeated for some reason e.g. disbelief, bugs, etc.*)
- Experimentation cycles (*describes an actual cycle of experimentation where one experiment lead to a new hypotheses and so on*)
- Overlapping experiments (*considerations for running many experiments in parallel*)

Paper IV

Data-Driven Software Design with Constraint Oriented Multi-Variate Bandit Optimization (COMBO)

Rasmus Ros Mikael Hammar

Abstract

Context Software design in e-commerce can be improved with user data through controlled experiments (i.e. A/B tests) to better meet user needs. Machine learning-based algorithmic optimization techniques extends the approach to large number of variables to personalize software to different user needs. So far the optimization techniques have only been applied to optimize software of low complexity, such as colors and wordings of text.

Objective In this paper, we introduce the COMBO toolkit with capability to model optimization variables and their relationship using constraints specified through an embedded domain-specific language. The toolkit generates personalized software configurations for users as they arrive in the system, and the configurations improve over time in relation to some given metric. COMBO has several implementations of machine learning algorithms and constraint solvers to optimize the model with user data by software developers without deep optimization knowledge.

Method The toolkit was validated in a proof-of-concept by implementing two features in collaboration with Apptus, an e-commerce company that develops algorithms for web shops. The algorithmic performance was evaluated in simulations with historic user data.

Results The validation shows that the toolkit approach can model and improve relatively complex features with many types of variables and constraints, without causing noticeable delays for users.

Conclusions We show that modeling software hierarchies in a formal model facilitates algorithmic optimization of more complex software. In this way, using COMBO, developers can make data-driven and personalized software products.

Keywords Continuous experimentation · A/B testing · Machine learning · Multi-armed bandits · Combinatorial optimization

1 Introduction

Design of user-facing software involve many decisions that can be optimized with user data. The decision variables—called the *search space*—can include both product aspects that are directly or indirectly visible to the user. For example, what wordings to use in headings or how items should be ranked in recommender systems [5] or search engines [286]. Traditionally, randomized controlled experiments (i.e., A/B tests or split tests) are used to iteratively validate the design choices based on user data [172]. Recently, data-driven optimization algorithms have been proposed to perform automated experimentation on software in larger scale on bigger search spaces simultaneously, at e.g., Amazon [138] and Sentient [208]. Personalization in particular is touted [138] as an opportunity to apply optimization algorithms to improve the user experience for different circumstances in, e.g., device types or countries.

The benefits of using optimization algorithms need to be balanced against the cost of implementing it. If the implementation cannot be broadly applied to many parts of the software product, then this investment might not pay dividends. Previously, data-driven optimization algorithms have only been applied to simple software [138, 209] with a flat structure in the decision variables, such as colors, layouts, texts, and so on. Software with more complex behaviors cannot be directly optimized with these techniques. We hypothesize that to handle more types of software the algorithms must understand the hierarchies that software is build with. For example, a software feature can have dependencies between variables such that one variable can only enabled if another one is, and so on.

We suggest modeling the search space and the relationships between variables—called *constraints* [25, 251]—in a formal language that can describe the software hierarchy. Developers can use constraints to exclude certain combinations from the optimization search space that would otherwise generate undesirable or infeasible variants. For example, the color of a button should not be the same as the background. Feature models [55, 159] from software product lines have been suggested by Cámara and Kobsa [48] as a suitable modeling representation to handle the variability of experimentation. With feature models, software variable dependencies are described in a tree hierarchy. Feature models also usually support a limited set of more complex constraints.

To this end we introduce the open-source toolkit called Constraint Oriented Multi-variate Bandit Optimization (COMBO) targeted at software engineers without deep optimization expertise. The toolkit consists of a domain-specific language for specifying a hierarchical model of the optimization search space with many types of constraints and multiple bandit-based machine learning algorithms (see Section 3) that can be applied to optimize a software product. To the best of our knowledge, this is the first attempt at combining bandit-based

optimization algorithms with constraints. We validate the toolkit’s capabilities in a proof-of-concept by implementing two feature cases relevant to the validation company Apptus, in e-commerce. The algorithmic performance is evaluated in simulations with realistic data for the feature cases.

Finally, we discuss the implications of using toolkits such as COMBO in the context of a data-driven software development process, which we define as the practice of *continuous optimization*. The current barriers to applying continuous optimization need to be lowered in order to encourage and enable developers to shift towards a higher degree of experimentation. For example, since modern software products are in a state of constant change, the optimization search space will have underperforming variables removed and new variables added. The algorithms need to gracefully handle such continuous updates of the search space model without restarting the optimization. We also call attention to several remaining barriers such as: handling concept drift [160] and ramifications on software testing [195]. We also provide considerations for what metrics could be optimized for and what the toolkit could be applied to.

The rest of this paper is structured as follows. Section 2 contains background and related work on continuous experimentation. Section 3 introduces theory on bandit optimization. In Section 4, the research context and methods are described along with threats to validity of the validation and limitations of the solution. In Section 5 the COMBO toolkit is presented. In Section 6, the toolkit is validated and the algorithms are evaluated. Finally, sections 7 and 8 discuss continuous optimization, metrics, future directions, and conclude the paper.

2 Background and Related Work on Continuous Experimentation

Many web-facing companies use continuous experimentation [106] for gauging user perception of software changes [9, 249]. By getting continuous feedback from users, software can evolve to meet market needs. Randomized controlled experiments (i.e. A/B tests, split tests, etc.) in particular are emphasized by high-profile companies such as Microsoft [164], Google [286], and Facebook [107] as an evidence-based way of designing software. The section below provides background on continuous experimentation through the lens of randomized controlled experiments for software optimization. This gives context to the main topic of our work on applying data-driven software optimization algorithms.

Experiments can be executed either on unreleased prototypes or after deployment to real users. Bosch-Sijtsema and Bosch [37] and Yaman et al. [318] explain how qualitative experiments on prototypes are used early in development to validate overarching assumptions about user experience and the business model. While post-deployment experiments, such as randomized controlled experiments, are used to measure small differences between software variants for optimizing a business related or user experience (UX) metric.

Prototype experiments are advocated for both in the lean startup framework by Ries [240] and in user experience research [52, 308]. Lean startup is about finding a viable business model and product through experimentation, for example, a pricing strategy suitable for the market. Lean startup has been brought to software product development in the RIGHT model by Fagerholm et al. [106]. Experiments based on design sketches are used within user experience to validate user interaction design, e.g., through user observations.

In both lean startup and user experience research there is a need to get feedback on prototypes early in the process. Though, as the product or feature design matures and settles, the shift can move towards optimization with randomized controlled experiments to fine tune the user experience. This can exist simultaneously with prototype experiments as different features in a product can have different levels of design maturity.

2.1 Randomized controlled experiments

In a *randomized controlled experiment*, variables are systematically changed to isolate the effect that each setting of the variable has on a certain metric. The variable settings are mapped to software configurations and each unique software configuration is assigned a user experiment group. When the experiment is deployed to a product environment each user of the system is randomly assigned to a user experiment group. Usually there are thousands of users per group to obtain statistical significance.

Randomized controlled experiments have been studied in data science as *online controlled experiments*. The tutorial by Kohavi et al. [172] from Microsoft provides a good introduction to the statistics and technicalities of it. The research includes studies on, e.g., increasing the efficiency [102] and trustworthiness of results [173].

The structure of the controlled experiment is referred to as the experiment design. In an A/B test there are two user experiment groups and in an A/B/n test there can be any number, but still with one variable changed. Thus, they are univariate. In a multi-variate test (MVT) there are also multiple variables that each can have multiple values. There are different strategies for creating experiment groups in an MVT. For example, in the full factorial design all interaction terms are considered so if there are n binary variables there would be 2^n groups. In a fractional factorial design only some interactions are considered.

Infrastructure is a prerequisite for controlled experiments on software [106]. The bare minimum is an experimentation platform that handle the randomized assignments of users and statistics calculations of the experiment groups. Microsoft have described their experimentation platform (ExP) for conducting experiments in large scale [130]. It has some additional optional features such as segmentation of users with stratified sampling for cohort analysis, integration with deployment and rollback of software, sophisticated alerting of suspected errors, and so on.

2.2 Personalization

MVTs are the current standard experiment design for having personalized experiment results [138, 172]. By *personalization* [108, 268] we mean that there are contextual variables that describe aspects of a user, such as device type or age, and that the point of personalization is to find different solutions for the different combinations of contextual variables. Having many personalization variables will result in needing many more experiment groups.

In the classical experiment designs of A/B/n tests and MVTs, users are allocated into experiment groups uniformly at random. That is, each experiment group will have equally many users. When the number of groups is large this can be inefficient. In any optimization algorithm, the allocation of users to experiment groups changes based on how well it performs in relation to some metric. Thus, they can concentrate users to the most promising configurations.

2.3 Experimentation implementation strategies

There are two distinct implementation strategies for randomized controlled experiments of software: using feature flags or multiple deployment environments. Firstly, feature flags [236] are essentially an if/else construct that toggle a feature at run time. This can be extended to do A/B testing. Secondly, having multiple software releases deployed to different environments, ideally done through containerized blue-green deployment [239]. The advantage of this approach is that the software variants can be organized in different code branches.

The number of experiment groups can be huge, especially with personalization and optimization. For the algorithmic optimization advocated in this work, having deployment environments for each combination of variable settings is infeasible. Thus, the feature flag strategy is presumed. However, there are scheduling tools that optimize the efficiency of experimentation in different deployment environments by Schermann et al. [262] and Kharitonov et al. [165].

2.4 Model-based experimentation and variability management

Experimentation introduces additional variability in software by design. Cámara and Kobsa [48] suggested modeling the variables through a *feature model* from software product lines research [159]. Feature models allow for the specification of hierarchies and other relations between feature flags and have been used to capture variability in many software products [55]. With feature models one can perform formal verification on the internal consistency of the model and perform standard transformations. This approach does not seem to have gained traction for continuous experimentation. Our approach of adding constraints and hierarchies (see Section 5) is not new per se, but the use of this in combination with optimization is novel to the best of our knowledge.

In practice, less formal methods are used for configuring many overlapping experiments at Google [286] and Facebook [15]. Facebook has open sourced parts of their infrastructure for this in the form of a tool [15] (PlanOut) that can be used to specify experiment designs. The tool also contains a namespace system for configuring overlapping experiments. In both companies' approaches, they have mutual exclusivity constraints where each experiment claims resources, for example, a specific button on a page. A scheduler or other mechanism ensures that experiments can run in parallel without interfering with each others' resources.

2.5 Automated experimentation with optimization algorithms

There is an abundance of tools that optimize parameters in software engineering and related fields, e.g., tweaking machine learning hyper-parameters [29, 275], finding optimal configurations for performance and efficiency [146, 142, 214], tweaking search-based software engineering parameters [8], and the topic of this work with software parameters for business metrics. In the various optimization applications, the assumptions are different on how the optimization problem is structured and what the technical priorities are. For example, when optimizing machine learning hyper-parameters for deep learning it is important to minimize the number of costly experiments. Bayesian optimization with Gaussian processes [275] is often used there, but the approach does not scale beyond a few thousand data points [241], because the computational cost depends on the number of data points for Gaussian processes.

One notable related research field is autonomic computing [163] that includes self-optimization of performance and efficiency related parameters, such as the loading factor of a hash table or what implementation to use for an abstract data structure. Such performance factors have relatively high signal-to-noise ratio in comparison to metrics involving users and the factors also exhibit strong interactions in terms of memory and CPU trade-offs. Many of these optimization tools (e.g. in [142, 214]) assume that the optimization is done

before deployment during compilation in a controlled environment. Some recent work has moved the optimization to run time and studied the required software architecture and infrastructure for cyber-physical systems [151, 121] and implications [196, 121] of this change.

We have also found several optimization approaches similar to our work, viz., they target large search spaces, with metrics based on many users, and are applied at run time in a dynamic production environment. The most similar work to ours is by Hill et al. [138] at Amazon where the problem formulation of multi-variate multi-armed bandits (see next section) is first formulated and addressed with a solution. There is also related work on search-based methods [148, 208, 285] and hybrids between search-based and bandit optimization [209, 250].

Search-based tools was first suggested by Tamburrelli and Margara [285] for automated A/B testing using genetic algorithms, and then independently by Iitsuka and Matsuo [148] for website optimization using local search. Ros et al. [250] suggested improving the genetic algorithms with bandit optimization and a steady state population replacement strategy. At Sentient they have implemented some of these ideas [250, 285] in a commercial tool with both genetic algorithms [208] and genetic algorithms with bandit optimization [209].

A *genetic algorithm* maintain a population of configurations and evaluate them in batches. The configurations are ranked by a selection procedure, and those that perform well are recombined with each other with some probability of additional mutation. This is implemented in our toolkit. However, they were not investigated further because we are not aware of an elegant way of supporting personalization with genetic algorithms. Maintaining a separate population for each combination of context variables is not feasible, because each separate population would have too few users. What we implemented in the toolkit was that when a user arrives with a given context, try to match it with a configuration in the algorithm's population, if there is no match then generate a new configuration with the selection procedure that is coerced to match the context and add it the population. This procedure scales better to more context variables due to concentrating effort on popular contexts. Though, it does not scale sufficiently to support the cases in Section 6.

3 Theory on Bandit Optimization of Software

Bandit optimization is a class of flexible optimization methods, see Figure 1 for a summary. Univariate bandit optimization is formalized in the multi-armed bandit problem (MAB) [47]. The name come from the colloquial term one-armed bandit which means slot machine. In MAB, a gambler is faced with a number of slot machines with unknown

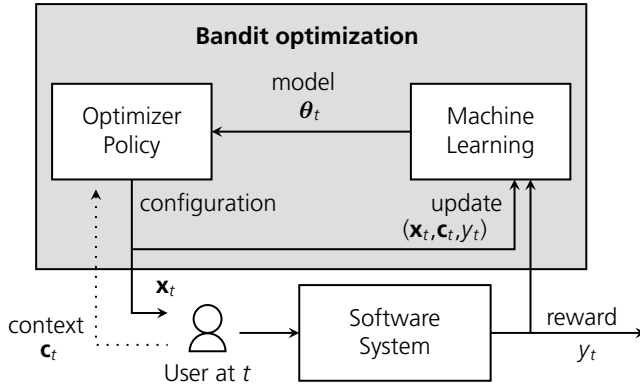


Figure 1: Bandit optimization setting summary. A user at time t of the system provides context \mathbf{c}_t and receives personalized configuration \mathbf{x}_t . The user provides the reward y_t by using the software system. An optimizer policy selects configurations which maximizes rewards based on a machine learning algorithm model θ_t . The model can predict rewards based on configurations and contexts and is continuously updated.

reward distributions. The gambler should maximize the rewards by iteratively pulling the arm of one machine. Applying MAB to A/B/n testing is sufficiently common to have its own name: *bandit testing* (such as in Google Optimize). The choice of arm is sometimes called an action but is referred to as configuration in this work.

An optimizer policy solves the MAB problem. Some of the policies are very simple, such as the popular ϵ -greedy. It selects a random configuration with probability ϵ , otherwise the configuration with the highest mean reward. A policy that performs well must explore the optimization search space sufficiently to find the best configuration. Policies must also exploit the best configurations that it has found to get high rewards. This is known as the *exploration-exploitation trade-off dilemma* [284] in reinforcement learning. In ϵ -greedy this trade off is expressed explicitly in the ϵ parameter. A high ϵ results in very random exploration, and a low ϵ results in almost pure exploitation of the best configuration, so it has a high risk of getting stuck with a local optima.

In the multi-variate case, the MAB setting is extended by a machine learning algorithm that attributes each reward to the variables in the multi-variate arm. This is known as multi-variate MAB [138] or more generalized as combinatorial MAB [58]. Contextual MAB is another more well known extension used for, e.g., recommender systems [186] which does personalization in the univariate case. In this work, we refer to all of these settings as bandit optimization, but the focus is on multi-variate MAB.

Herein lies the crucial difference between the multi-variate MAB setting and other forms of black-box optimization (where the objective function is unknown). Namely, that the algorithm learns a *representation* in the form of an online machine learning model and optimizes it with respect to its inputs. Some black-box optimization algorithms [8, 142, 214] find and keep track of the best performing data points and iteratively replaces them

with better performing ones. That is not the case in our work because of the assumption that the signal-to-noise ratio is so low that estimating the performance of a specific data point is inefficient. Other black-box optimization algorithms [146, 241, 275] have offline machine learning models that need to keep track of all data points and retrain the machine learning model at intervals, that does not scale well to very large data sets.

To summarize so far, a more precise definition of bandit optimization follows. A user arrives in the system at time step t . A software configuration $x_t \in \mathcal{X}$, $x_{t,i} \in \mathbb{R}$, $i = 1, \dots, n$, is chosen for the user with context variables $c_t \in \mathcal{C}$, $c_{t,j} \in \mathbb{R}$, $j = 1, \dots, m$, where n is the number of configuration variables, m the number of context variables, \mathcal{X} is the search space of decision variables, and \mathcal{C} is the context space. When the user is no longer using the software system, a reward $y_t \in \mathbb{R}$ in some metric is obtained from the user. A policy is tasked with selecting x_t such that the rewards are maximized over an infinite time horizon.

3.1 Practical implications on software systems

When applying bandit optimization in practice there are multiple assumptions:

- The number of users are at least measured in the thousands. Depending on how many variables are included in the search space and the signal-to-noise, the required number of users will be more or less.
- The value incurred by a user must be quantified to a single value at a discrete time. This will be a simplification because users continuously use software.
- The optimization is done online at run time with continuous updates. A user's configuration must be generated quickly, so that users do not suffer noticeable delays. For software with an installation procedure (desktop or mobile) the optimization can be done during this installation.
- The rewards are assumed to be identically and independently distributed for all users. The algorithms are not *guaranteed* to perform well when the reward distributions change over time, though they still might perform well.

Additionally, in comparison to A/B testing, bandit optimization requires storing the configuration per user. Standard procedures for controlled experimentation assign experiment groups with a pseudo-random allocation with a random seed based on something that is static and unique for the user, e.g., a user's id. As such, a user will have the same experiment group even if their session expires. With bandit optimization, they might be assigned to another group unless the group assignment is stored persistently, until the user's session is completed. Persistent storage of user group assignment has implications on data privacy [133].

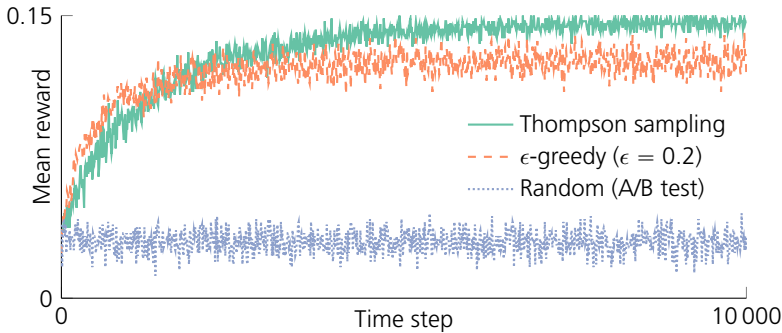


Figure 2: Example simulation of different bandit policies with binary rewards. There are five available arms, four with mean reward 0.1 and one with reward 0.15. The simulation is repeated 100 times and the plot shows the average result. The figure illustrates that Thompson sampling eventually converges to the optimal reward while ϵ -greedy converges to $0.1\epsilon + 0.15(1 - \epsilon) = 0.14$.

Mattos et al. [200] conducted an empirical investigation into bandit testing for software. They found that while the technique performs better at finding good configurations, it can lead to statistical issues such as inflated false positive error rates. They advice to apply bandit testing only when the consequences of false positives are low, as it is with optimization.

3.2 Thompson sampling

The optimizer policy of choice in our work is Thompson sampling since it often performs better than ϵ -greedy in general [54, 256]. The idea is to match the probability of selecting an arm with the probability of it being optimal. The procedure is as follows for standard MAB. A Bayesian posterior distribution is placed on the expected reward of each arm. In each step t , a sample $\hat{\theta}_{k,t}$ is drawn from each posterior distribution $P(\theta_k | \mathbf{y}_k)$, where k is an arm index, θ_k is the prior reward parameter of the respective arm, and \mathbf{y}_k its previous rewards. The arm a_t at step t with the highest sample is selected, that is $a_t = \arg \max_k \hat{\theta}_{k,t}$. The posterior distribution is updated continuously.

For example, if the rewards are whether users click on something or not. Then, a suitable posterior distribution family for binary rewards is the Beta-distribution, parameterized by the number of clicks and non-clicks. Figure 2 shows a simulation of this, comparing Thompson sampling, ϵ -greedy with $\epsilon = 0.2$, and random (as in standard A/B/n testing). Thompson sampling and ϵ -greedy initially behave like the random policy, but improve quickly. Thompson sampling finds the optimal arm, while ϵ -greedy is stuck with a random arm with probability 0.2.

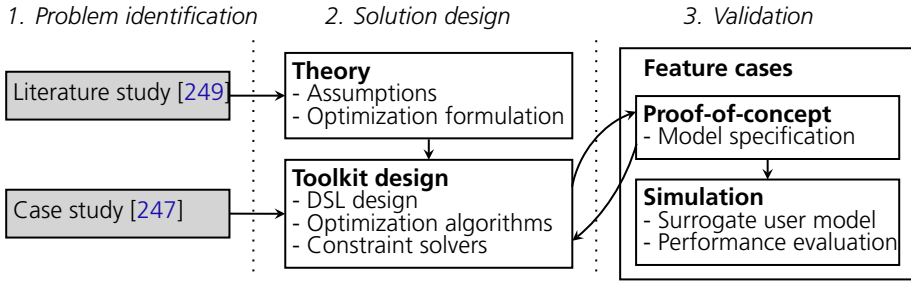


Figure 3: Research methods overview divided in three stages. Based on previous studies (gray boxes), the theory was identified and the design of the toolkit was designed to support the validation company’s product. The toolkit was evaluated with two feature cases, that were first implemented in the toolkit and then subject to simulations.

For multi-varate MAB the posterior distribution is a probabilistic machine learning method denoted q parameterized by the machine learning model θ . A single joint sample $\hat{\theta}_t$ is drawn from the multi-variate distribution $P(\theta|y)$ each step. The configuration x_t is then chosen as such: $x_t = \arg \max_{x \in \mathcal{X}} \mathbb{E}_{q_{\hat{\theta}_t}}[y_t | x_t = x, c_t]$. Depending on the machine learning method q this optimization problem is set up differently and it will be significantly harder to solve than in the univariate case. In MAB, the computational complexity at each step is linear in the number of arms, the $\arg \max$ calculation can be a for-loop over each sample. While for multi-variate MAB, the computational complexity at each step is NP-hard. Section 5.2 contains the specifics for how we did the setup of the optimization for different machine learning methods in the toolkit.

4 Research Context and Methods

Our research was an on-site collaboration with the e-commerce company Apptus spanning 20 weeks, and was part of a long term research project. The research was conducted with a design science methodology, following the guidelines by Wieringa [306] and Runeson et al. [255]. As such, the toolkit was designed as a solution that addresses an industrially relevant problem, and the validation of the solution is done with sufficient rigor (see Figure 3 for an overview). This section ends with discussing the limitations of the solution design and threats to validity of the validation.

4.1 Validation company and e-commerce

The validation company Apptus is a small Swedish company which develops a platform for e-commerce. Their product platform provides web shops with various data-driven algorithms. It includes a recommender system, product search engine, ads display algorithms, etc. Apptus deploy their software to other companies’ web shops, so they have a business-

to-business relationship (B2B) with their customers. Apptus has no direct relationship with the end-users of the software (consumers), but have access to consumer data through their customers' web shops. Operating multiple web shops incur a greater need for personalization (see Section 2.2) to optimize their software for different circumstances.

Experimentation is well established in e-commerce for a number of reasons; we believe this is the case for four reasons. First, the consumers often have a clear goal in mind to purchase a product. Second, this goal is aligned with the goals of the web shop companies. Third, there is an industry standard for quantifying this joint goal through the sales funnel of: clicks, add-to-carts, and purchases. Finally, consumers tolerate some degree of change in the interface, especially in what ads and products are displayed.

A prior case study by Ros and Bjarnason [247] outlines how Apptus uses continuous experimentation to improve their product in several scenarios: validating that a change has the intended outcome, manual optimization, and algorithmic optimization. Thus, Apptus is experienced with using optimization in various forms to optimize the web shops. They also use bandit optimization in their product recommendation system [40] and as part of a customer facing experimentation platform (targeted at marketers and software engineers) that optimizes which algorithm should be active in which parts of the web shop.

4.2 Research stages

The design and validation of the COMBO toolkit was done in three stages (see Figure 3): (1) *identifying a problem* with industrial relevance through a literature study [249] and a case study with interviews [247], (2) *designing a solution* to solve the problem, and (3) *validating* that the solution works.

4.2.1 Problem identification

Prior to this study, two studies were performed for problem identification. First, a *systematic mapping study* [249] on continuous experimentation identified suitable algorithms to the problem domain and the assumptions in the optimization formulation as stated in Section 3. Second, an *interview study* with five participants from the validation company was conducted on scenarios that experiments are used in [247], and automated optimization was one such scenario. One prominent challenge with optimization, was that the algorithms are specific to a certain circumstance (e.g., product recommendations) and are hard to apply outside these circumstances. This challenge was also present in related work in their narrow application to visual design and layout.

4.2.2 Solution design

The design of the toolkit was done in iterations with subsequent validation, to ensure that the toolkit had the necessary functionality to support the validation. The decisions taken in the design include what optimization algorithms to implement, what constraints to support, and how the search space should be specified. The design choices were anchored in two workshops with employees at the validation company. The approach was to be inclusive in terms of optimization algorithms by using available optimization libraries. The constraints supported were simply added as needed. The search space and constraints specification was a choice between meta-programming with annotations as featured in some related work [142, 285] and declarative code with a domain-specific language, where the latter was chosen because it is more flexible for users of the toolkit.

4.2.3 Validation

The toolkit was validated on-site at the validation company by two *feature cases*. Each feature case include a validation proof-of-concept demonstration and evaluation simulations. Technical details on the simulation set up is given in Section 6 alongside the presentation of the feature cases. The first feature case, an auto complete widget, was identified in a brainstorming workshop with three employees from the validation company. It was chosen because it has both a graphical interface and is a data-driven component, while still being a well isolated part of the site. The second feature case, a top- k categories listing, was selected in a discussion with an employee to push the boundaries of what is technically feasible with the toolkit and because it was an existing feature that had real historic user data.

The proof-of-concept validation was conducted to demonstrate the toolkit's soundness. It included a step to ensure that all necessary variability of the feature case was captured and then demonstrating that it could be implemented in the toolkit. For the auto complete feature we found a listing of top 50 fashion sites and filtered it down to 37 fashion web shops that had an auto complete widget. We analyzed how the widgets varied and then implemented variables and constraints to sufficiently capture the variability. In addition, there was a brainstorming workshop with a user experience designer to validate the choices. For top- k we chose a client web shop that had a sufficient number of users and a complex category tree and re-implemented the functionality of the original top- k categories.

Simulations of the feature cases were performed to show that the optimization algorithms can navigate the search space in ideal circumstances, within a reasonable data and execution time budget. The use of simulation avoids the complexities of a deployment and the evaluation can be repeated and reproduced as many times as needed. Also, it enables benchmarking the different algorithms. In the simulation, the optimization algorithm iteratively

chooses a configuration from the search space and a reward from the user is simulated to update the algorithm. However, the historic user data could not be used directly to simulate users, because all combinations of the variables in the model are not present in the data. Instead we used the historic data to train a supervised machine learning model that can predict the expected reward of each configuration. This prediction can be used to simulate user rewards from any configuration. The machine learning model is an instance of a *surrogate model* [117], which is a general engineering technique used as a replacement for something that cannot be directly measured, which are users in this case.

4.3 Data collection

Both qualitative research notes and quantitative data sets for evaluation were collected. Decisions taken during solution design and implementation of the feature cases were recorded as notes, as recommended by Singer et al. [273]. The notes were consulted during the write up. The notes were of two types. First, in the form of 20 weekly diaries when on-site at the validation company. They contained notes on decisions taken and considered during solution design, and implementation of the feature cases. Second, notes were taken after the three workshops outlined above, (1) when identifying the first feature case, (2) when presenting the toolkit design choices, and (3) when brainstorming the variability of the first feature case.

The data sets used to train the surrogate user model were collected at the validation company. The second feature case (the top- k categories) had historic data that was used. For the first feature case (the auto complete widget) we collected data through a questionnaire at the validation company. There, we collected screenshots from the 37 fashion web sites with an auto complete widget. Then, 16 employees were asked to rate 20 randomly chosen screenshots on a 10 point Likert scale to rate the user experience of the auto auto complete widget. Many of them mentioned that it was hard and somewhat arbitrary. However, the data set was only used to get a somewhat realistic benchmark with actual interactions between variables, in comparison to a completely synthetic benchmark with randomly generated data.

4.4 Limitations

There are technical and statistical limitations to what is possible with the toolkit. The number of variables that can be solved for within a given millisecond budget is limited by the number of constraints, how sparse the configuration is (i.e. how many variables are, on average, zero valued), and the complexity of the underlying machine learning model. The

ability to make inference from the learned model will also depend on the machine learning model complexity, for example, inference from neural networks is notoriously difficult. This limitation is a fundamental trade-off between algorithmic performance and inference ability.

There are also limitations on the impact that can be obtained from applying the optimization. In e-commerce, much of the interface can be measured directly in terms of its impact on revenue, thus the ability to have an effect is promising. We believe the toolkit can be of use also outside of e-commerce, although there must be a way to quantify the user experience or business value of a given feature.

Finally, we are not claiming that all experiments in software engineering should be replaced by optimization. For instance, when adding optimization capabilities to a software feature, it would be prudent to validate that the optimization actually works with an ordinary controlled experiment.

4.5 Threats to validity

Here threats to validity that threaten the validation are identified along with the steps taken to mitigate them.

The *external validity* of the feature cases are threatened by that the evaluation is performed with only one company. This is mitigated by that the validation company has multiple clients that operate web shops. Thus, the feature case implementations are relevant to multiple company contexts, though still within the e-commerce domain.

The *internal validity* of the validation is dependent on the quality of the data sets and the resulting surrogate user models. If the surrogate user models are too easily optimized by an algorithm, then the simulations will be unrealistic. For example, if there are regions in the optimization search space with low data support the surrogate user model might respond with a high reward there. To mitigate this threat we evaluated the performance with a baseline random algorithm and an oracle algorithm with access to the surrogate user model. In this way, an upper and lower bound of reasonable performance is clearly visible.

Also, both the external and internal validity of the evaluation would be increased if the toolkit was evaluated in a production environment in a controlled experiment. A deployment to a production environment would uncover potential problems that cannot be seen through simulations. Thus, an actual deployment to real users is a priority in future work. However, the approach with surrogate user models for simulations is sufficient to demonstrate the toolkit's technical capabilities. The use of the surrogate user models also increases *replicability*, since at least the simulations are fully repeatable (see Section A).

5 Tooling Support for Bandit Optimization

The open source toolkit Constrained Online Multi-variate Bandit Optimization (COMBO)¹ is a collection of bandit optimization algorithms, constraint solvers, and utilities. This section will first present concepts in the toolkit and then specifics of the included algorithms. COMBO is written in Kotlin and can be used from any JVM language or Node.js. Kotlin is primarily a JDK language but can transcompile to JavaScript and native with LLVM.

To use the toolkit one must first specify a search space of variables and constraints in an embedded domain-specific language (DSL) and map them to software configuration parameters. *Variables* can be of different types: boolean, nominal, finite-domain integers, etc. Though COMBO is optimized for categorical data rather than numeric data. For example, the internal data representation is by default backed by sparse bit fields. *Constraints* can be of types: first-order logic, linear inequalities, etc. For example, if one boolean variable b requires another variable a the constraint $b \Rightarrow a$ can be added.

As mentioned in Section 3, some use cases of the toolkit require context variables for personalization. There is no explicit difference between decision variables and context variables in COMBO—any variable can be set to a fixed value when generating a configuration and have the rest of the variables chosen by the algorithm.

Variables and constraints are always specified within a *model*. There can be any number of nested models in a tree hierarchy. The hierarchy is inspired from the formal variability management language of *feature models* [48]. The hierarchy fulfills two additional purposes. First, having a mechanism for the case when a variable has sub-settings but the variable itself is disabled; then the sub-settings should not be updated. It is counted as a missing value by the bandit optimization algorithm. A variable can also be declared as *optional* in which case it has an internal indicator variable that specifies whether it is missing. Second, the hierarchy supports lexical scope to enable composability. That is, models can be built separately in isolation and then joined together in a joint superordinate model without namespace collisions, because they have different variable scopes.

Each model has a proposition that governs if the variables below it are toggled. The constraint $b \Rightarrow a$ can also be expressed implicitly through the tree hierarchy if a is the root variable and b is a child variable, as such:

```
model("a") { bool("b") }
```

The example shows a basic model of a search space with a variable b and the root variable a . It has the implicit constraints $b \Rightarrow a$ and a with two solutions: (a, b) and $(a, \neg b)$. Note that the root variable is always a boolean which is also added as a unit constraint.

¹The source code is available at <https://github.com/rasros/combo>.

```

model("Auto complete") {
  val width = nominal("Width", "S", "M", "L")
  optionalNominal("Search suggestions", 1, 3, 5, 10)
  val cards = optionalNominal("Product cards", 1, 3, 5)
  model(cards) {
    bool("Two-column")
    val hz = bool("Horizontal")
    impose { width["S"] equivalent !cards }
    impose { width["L"] equivalent (hz and !cards[1]) }
  }
  impose { "Search suggestions" or "Product cards" }
}

```

Figure 4: Partial model specification for the auto complete search widget (see Section 6.1). The example illustrates: the hierarchy with a child model, different types of variables: *bool* or *nominal*, optional variables, and imposed logic constraints.

Figure 4 shows a more advanced example, there are five variables and three of them are defined in the root model. The variable *Product cards* is the root of a sub model and is the parent to the variables *Two-column* and *Horizontal*, so there are implicit constraints $Two\text{-}column \Rightarrow Product\ cards$ and $Horizontal \Rightarrow Product\ cards$. Conceptually, in the application that uses COMBO, the variables *Two-column* and *Horizontal* modify the behavior of the *Product cards* so the first two variables can only be true when *Product cards* are enabled.

Further details of the DSL illustrated by the example in Figure 4 are summarized below:

- Non-root models do not need to declare new variables, they can use any proposition (constraint or variable) as its toggle.
- Explicit constraints are added using an *impose* block. They can be first-order logic, linear inequalities, cardinality constraints, and reified constraints.
- In constraints, variables are referred to by their name as a string within their lexical scope or through an ordinary object reference.
- The indicator variable of optional variables and the specific values of a variable can be used in constraints, as is seen in the usages of the variable *Product cards*.

5.1 Constraint solvers

The model specifies a constraint satisfaction problem which is solved with, e.g., finite-domain combinatorial constraint programming solvers [251] or boolean satisfiability (SAT) solvers [25], depending on what type of constraints and variables are used. The primary use of the constraint solvers is to perform the arg max calculation of multi-variate Thompson sampling, as part of the Optimizer Policy box in Figure 1.

We see three more uses for constraint solvers for experimentation that are enabled by the toolkit. First, a constraint solver can be used to formally verify the model, which is the main point of Cámara and Kobsa [48]. For example, by verifying that each variable can be both enabled and disabled. Second, randomly ordered permutations of configurations can be used to sample the search space for integration testing purposes. Third, in a large scale MVT with a fractional factorial design (see Section 2.1) and constraints between variables, being able to generate random solutions is required. The problem of generating random solutions uniformly is known as a witness [51].

The toolkit includes the SAT solver Sat4j [23] and constraint programming solver Jacop [180]. It also features two search-based optimization algorithms: genetic algorithms and local search with tabu search, annealing random walk, and unit constraint propagation [157]. The extensions to local search are crucial because there are mutual exclusivity constraints created through nominal variables that would be otherwise hard to optimize with.

When used in combination with the bandit optimization algorithms listed below the solvers should optimize some function rather than just deciding satisfiability. The specifics of how this is done depends on the bandit algorithm. In general, both the black-box search-based methods and the constraint programming solvers can be applied to any function, while the SAT solvers can be applied to optimize linear functions with integer weights if they support MAX-SAT.

5.2 Bandit optimization algorithms

The sections below give some intuition behind how the multi-variate bandit optimization algorithms included in the toolkit function. They include decision tree bandit [62, 97], random forest bandit [110], generalized linear model bandit [54, 138, 186], and neural network bandit [241]. All of them have fully custom implementations in COMBO, except the neural network bandit which is partially implemented using the Deeplearning4j framework.

5.2.1 Decision tree and random forest bandit.

Decision trees recursively partition the search space into homogeneous regions. It has been formulated as a contextual MAB algorithm with Thompson sampling [97] and used for A/B testing in practice [62]. The algorithm has a tree for each arm that partition the contextual variables. The trees are updated iteratively using the VFDT [84] procedure where each tree initializes with a root node and adds splits greedily. When selecting an arm the statistics of the leaf node corresponding to the user context is used by an optimizer policy.

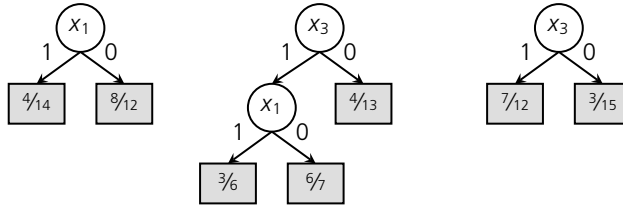


Figure 5: Example of random forest bandit with three trees. The square leaf nodes show the bootstrapped statistics for binary success/failure. Statistics for potential splits are also kept at each leaf node but not shown here.

When adapting this to multi-variate bandit optimization we use only one tree that splits both decision variables and context variables. Due to the partitioning of the search space the posterior distributions used for Thompson sampling can be defined separately.

The policy of selecting a software configuration from a given tree with Thompson sampling can proceed as follows in three steps. First, sample a value from the posterior distribution of all leaf nodes and select the leaf node with the maximum sampled value. Second, calculate the configuration from the leaf node by following the leaf node to the root node and set all parameters according to the splits taken in the path. Third, any x -value unset in the selected leaf node can be chosen at random. For example, consider the middle tree in Figure 5 with root split on x_3 , and then further split on x_1 for $x_3 = 1$, with $\mathbf{x} \in \{0,1\}^3$. If the leaf node where $x_3 = 1$ and either $x_1 = 0$ or $x_1 = 1$ is selected, then x_2 can be randomly selected. For the leaf node where $x_3 = 0$, both x_1 and x_2 is randomly selected while satisfying the constraints.

Random forest is an ensemble learning version of multiple decision trees that often perform better than an individual tree. Each tree sees a random subset of variables and data points. Random forest also has a contextual MAB formulation [110], although the derivation of this work to multi-variate MAB is unclear. Instead, we did a monte carlo tree search [42] over time by augmenting each decision tree in the ensemble with statistics at each split node that aggregates the data of all children below it.

When selecting a configuration from the random forest the following procedure is applied iteratively. Aggregate all split nodes at the top level of each decision tree by their decision variable. Sample a value from each decision's pooled statistics and select the best split. Then update the top level node of each tree by following along the split decision. Continue until all trees are exhausted.

Consider the example random forest in Figure 5. The posterior distributions for the top level decisions are: $x_1 = \{0,1\}$ with Beta(8,12) versus Beta(4,14) and $x_3 = \{0,1\}$ with Beta(3.5,14) versus Beta(8,12.5). Note here that the statistics for the middle tree's x_1 node is not counted in the x_1 decision since that node is for x_1 conditioned on $x_3 = 1$, i.e., $P(\theta_1 | y_1, x_3 = 1)$. Then, four samples are drawn from the distributions and the decision corresponding to the

highest sample is selected. Suppose $x_3 = 1$ is selected, then the left tree is unchanged, the middle tree will have a new top node x_1 and the right tree is exhausted. The next decision is only on x_1 with distributions Beta(3.5,10) versus Beta(7,9.5), after that decision all trees are exhausted.

5.2.2 Generalized linear model and neural network bandit.

Bandit optimization with linear models for contextual MAB have seen lots of use for recommender systems [54, 186] and have been adapted to multi-variate bandit optimization at Amazon [138] with Thompson sampling. Any type of regression that fits in the generalized linear model (GLM) framework can be used, such as logistic regression for binary rewards or linear regression for normal distributed rewards.

Each variable in the search space and context space has an estimated weight $\hat{\theta}$. The predicted value is a linear combination of the weights $\hat{\theta}$ and the concatenation of variables x_t and context c_t . For measuring the uncertainty of a prediction we also need a continuously updated error variance-covariance matrix Σ of the weights. For ordinary linear regression models, Σ is $\sigma^2 (X^T X)^{-1}$, where σ is the standard error and X is a matrix where each row is a user's configuration. As more data points are added the values in the covariance matrix will shrink. For generalized linear models, the Laplace approximation [54, 256] yields a similar calculation.

The linear model is updated continuously with second-order stochastic gradient descent using the covariance matrix updates as such²:

$$\begin{aligned}\Sigma_t^{-1} &= \Sigma_{t-1}^{-1} + \nabla^2 g_t(\hat{\theta}_{t-1}), \\ \hat{\theta}_t &= \hat{\theta}_{t-1} - \Sigma_t \nabla g_t(\hat{\theta}_{t-1}),\end{aligned}$$

where ∇g is the gradient of the prediction error of a specific input vector (representing a software configuration) at time step t .

With this setup Thompson sampling can be used as a policy as follows. Generate a sampled weight vector $\tilde{\theta}_t$ from the multi-variate normal distribution $\mathcal{N}(\hat{\theta}_t, \Sigma_t)$, where $\hat{\theta}_t$ are the model weights and Σ_t is the error variance covariance matrix of the weights. Subsequently, the action x_t chosen at step t is the one which maximizes the expected rewards from the sampled weights using the linear prediction, i.e., $x_t = \arg \max_{x \in \mathcal{X}} \tilde{\theta}_t^T(x, c_t)$, subject to the constraints.

²Russo et al. [256] present a tutorial on how to make the calculations online efficiently with quadratic complexity to the number of variables. A simplified diagonalized covariance is often used instead for its linear complexity and ability to exploit sparseness in the decision variables [54, 138, 186].

This objective function can be solved efficiently with many approaches. Including search-based methods which Hill et al. [138] used, constraint-programming, or either of MAX-SAT and mixed integer linear programming solvers if the constraints are limited to first-order logic and linear inequalities respectively.

Neural networks has been applied to bandit optimization as well. Riquelme et al. [241] did a simulation evaluation of various deep learning and linear bandits for contextual MAB. The linear model bandit mentioned previously and a version called *neural linear* were the winners of most experiments. In neural linear, a neural network is fed into a linear bandit where the uncertainty is estimated in the linear model and the network is used for improved representation of the structure of the search space.

Having a neural network be continuously updated can be done in many different ways. In COMBO, the linear model in neural linear is updated continuously on every step but the network is updated in batches. The batch update is kept for some configurable time and used to train the neural network in multiple epochs, after which it is discarded.

The objective function is significantly harder for the neural networks than for GLM. Constraint-programming is possible but it will be very slow due to poor constraint propagation through the network. The only appealing option in the toolkit is the search-based methods.

6 Validation

The COMBO toolkit was validated by implementing two feature cases and then evaluated by simulating their usage as realistically as possible using data from the validation company. One of the cases modifies an existing feature to make it data-driven and the other improves and generalizes an existing algorithm. As detailed in the Section 4.2.3, the procedure was done in four steps as follows. First, the variability of the feature case was analyzed. Second, the variability was implemented in the toolkit as a proof-of-concept validation. Third, a data set was collected with configurations and reward measurement pairs and a surrogate user model was trained using the data set. Finally, the surrogate user model was used to repeatedly simulate users in a controlled environment. The simulation results are summarized jointly in Section 6.3 and replication instructions are available in Section A.

6.1 Feature Case 1: Auto complete widget for product search

The validation company has recently expanded to provide graphical interfaces to their algorithms, targeted at fashion web shops. One of the algorithms with a new graphical interface that they provide is the auto complete search that pops down when a user starts typing a search query. The case was chosen in a workshop session for two reasons. First, there is

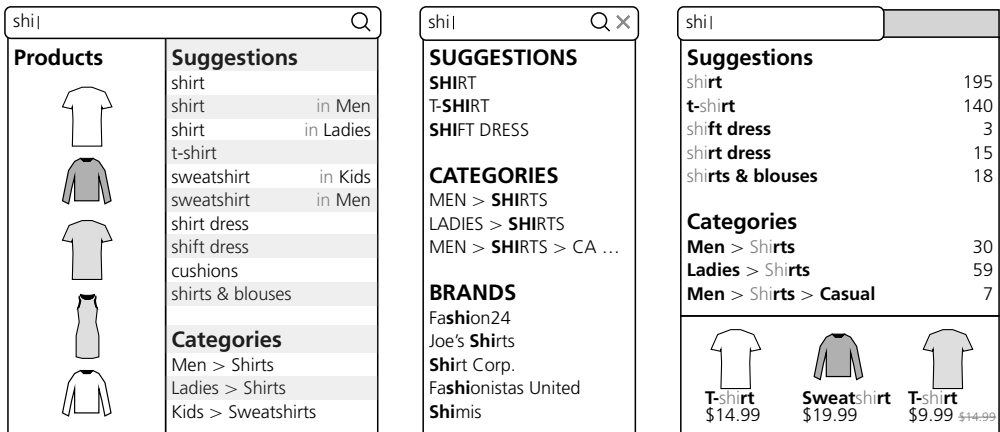


Figure 6: Visualization of different auto complete widgets. To the left are two randomly generated designs and to the right is one with a high score. The scores of the surrogate user model are in increasing order from left to right, with scores: -0.02 , 0.01 , and 0.31 , where higher scores indicate better perceived user experience. All were generated with constrained equal height for illustration purposes.

no industry consensus on how an auto complete widget is supposed to look. There is not even an agreement on what type of items the widget should show: suggested search terms, brands, product categories, or products. It also might be the case that different web shops require different configurations depending on available product data and how big and diverse the product catalogue is. Second, it is an isolated component with low risk that does not affect the rest of the site appearance.

6.1.1 Model variability

There were three inputs to the decisions taken for modeling the auto complete feature. First, a brainstorm session with a user experience designer was conducted. During the session, several existing auto complete widgets were investigated. The optimization target was decided to be usability, such as whether or not the suggestions saw use or not. It was decided to include only user experience variables and exclude visual design parameters from the model. This was because the validation company preferred if the visual design was consistent with the web shops' general style. Examples of user experience variables are: the types of data to use, whether to have two columns, whether prices and/or sales indicator should be shown with images, etc. Example design parameters are: colors of highlight and borders, font selection and size, etc.

Second, we used guidelines produced from a company specialized in user experience (UX) research for e-commerce: Baymard Institute. They have conducted user tracking sessions and synthesized advice for auto complete and more. The guidelines are not free and the specifics cannot be disclosed, other than that as a requirement from the validation company the guidelines should be adhered to in the designs, either as a decision variable in the optimization or as a constraint. It included things like the maximum number of items to display, how to keep the different data types separate, and some common specific pitfalls.

Third, screenshots from 37 fashion web sites were analyzed to serve as test cases such that the model captures all the salient variability on the sites. Only variables with five or more occurrences were included in the model.

6.1.2 Model implementation

The resulting model of the auto complete in the COMBO DSL can be seen in a much simplified form in Figure 4. Some visualizations renditions of the model are given in Figure 6. The full model has 32 variables (or 53 binary variables), with 28 explicit constraints and 43 implicit constraints from hierarchy or variable specification. Some relevant parameters not visible in Figure 6 are: underlined highlight, strict search matching, images with fashion models, etc. Among others, there are constraints to calculate the total discretized width and height of the widget (which can be used to generate widgets of specific dimensions), constraints for exclusive variables (e.g., in Figure 6 the inlined categories that say *in Men* to the left and the counts to the right occupy the same space), and that there must be at least either search term suggestions, product cards, or category suggestions.

After the data was collected it took 10 hours to construct the model by the first author. Having test cases available made the process of eliminating invalid variants much easier, such that some valid combinations were not accidentally removed. Also the ability to do formal validation was useful, by querying the system for whether a specific combination was valid or not.

6.1.3 Surrogate user model implementation

Since the functionality of the auto complete widget was new at the time, there was no data to base the surrogate user model on for the simulations. Instead a questionnaire on the 37 collected sites were constructed; for each web shop a search on the ambiguous search term 'shi' was conducted. Then participants were asked to score the usability of 20 randomly chosen web shops on a 1–10 scale. In total 16 participants completed the questionnaire which resulted in 320 data points.

A data set was then constructed by describing each web site in terms of the model. The score was z-normalized to mean 0 and variance 1, as such, the left most widget in Figure 6 is below average while the middle is slightly above. The data set was used to train the surrogate user model. Linear regression was chosen since there was not enough data points for anything more sophisticated. Generating a simulated reward for bandit feedback was then done by making a prediction with the surrogate user model and adding noise estimated from the standard error of the fitted model.

Variables were added to handle the following confounding factors: persons scoring the web site, whether suggested terms and images were relevant to search, whether there were duplicated suggestions, what genders the search results were targeted to (male, female, mixed, and unisex). In addition, some pairwise interaction terms were added. These extra variables were not part of the search space but they improved the predictive power of the surrogate user model. The final surrogate user model size was 141 binary variables and 141 constraints (coincidentally equal).

6.2 Feature Case 2: Top- k categories

Many web shops have organized their product catalogue as a category tree (c.f., Figure 6 under the Categories headings). The validation company provides many algorithms related to the category tree; one example is displaying a subset of the top- k most relevant categories. Where k is the number of categories to display in a given listing. A naïve algorithm would be to simply display the most clicked categories, this might then not sufficiently cover all users' interests. They have more advanced versions in-place already in Hammar et al. [134]. The purpose with this feature case is to show that a generic implementation of the feature is possible using the toolkit. Another goal was to evaluate the algorithms with a more challenging optimization problem than the previous feature case. The search space is much larger and since the data is from real usage the signal-to-noise ratio is lower.

6.2.1 Model variability

Since the top- k feature is a re-implementation of an existing feature we could select a real web shop and use their category tree. The chosen web shop had sufficient data volumes and a category tree of medium size. The category tree has 938 nodes, with a maximum depth of 4. The web shop that the data came from operates in three different countries. The country that a user comes from was added as a nominal variable to the model. In the simulations the country variable was used for personalization and was generated randomly.

```

model("Category tree") {
  model("Ladies") {
    bool("Top level")
    model("Jeans") {
      bool("Top level")
      bool("Skinny")
      bool("Loose")
      impose { "Jeans" equivalent or(scope.variables) }
    }
    model("Shirts") { /* More categories follows */ }
    impose { "Ladies" equivalent or(scope.variables) }
  }
  val k = int("Top-k", min = 1, max = 100)
  impose { cardinality(k, LE, leafCategories()) }
}

```

Figure 7: Partial model specification in COMBO for the top- k categories feature. The function *leafCategories* has its implementation omitted, it should return each variable that does not have any sub variables.

6.2.2 Model implementation

The model corresponds directly to the category tree, see a simplification in Figure 7, the actual model is programmatically built different for each specific web shop. A constraint is added to the bottom of each sub model to enforce that one of the sub models' variables are active. In addition to the constraints from the hierarchy, there is also a numeric variable k that control how many categories there can be. A *cardinality* constraint ensures that the number of categories are less than or equal to k . The k -variable can be set for each user or chosen by the decision algorithm. The total model size is 1 093 variables with 1 101 binary values and 1 245 constraints.

6.2.3 Surrogate user model implementation

The data set for the surrogate user model was collected with 2 737 568 configurations and reward pairs. Each data point was constructed by observing what categories were clicked on for each consumer and a reward of 1 was received if the consumer converted their session to a purchase and 0 otherwise. Only data points with at least one click on a categories listing were eligible. In this case, the parameter k is derived from historic user data in the training set for the surrogate user model. This means that the data set was collected for a slightly different scenario than what it is used for. One artifact of this is that the learned

surrogate user model is maximized by having $k = 100$, since that correlates with users that spend more time on the site and are likely to convert to customers. However, this effect would not be present in the actual use case. To counteract this, the k parameter is fixed to a specific value ($k = 100$) during simulation.

Since there were lots of data and the point was to make a challenging problem, a neural network was chosen to be the surrogate user model. A standard feed-forward neural network was trained with PyTorch for 30 minutes with desktop hardware. The network topology is: first a dropout layer; then two hidden layers with 15 nodes each, ReLU activation, and batch normalization; and finally a softmax output.

6.3 Simulation evaluation results

All bandit optimization algorithms from Section 5.2 and two baselines were evaluated on both surrogate user models, see an overview of the results in Table 1. Again, these results do not provide strong evidence in favor of one algorithm or another, but they show the feasibility of the approach and highlight important choices in algorithm design. Local search was used as optimizer by all the algorithms with the same configuration, except neural linear which was tuned to improve the speed at the expense of performance. The simulations were done on the JVM with desktop hardware with an 8-core computer.

All simulations in the table and figures were repeated for multiple *repetition runs*: 1000 runs for auto complete and 200 runs for top- k categories. In each repetition, each algorithm start from a blank slate with no learned behavior. The algorithm iteratively chooses a configuration and updates the underlying machine learning model at each time step, for time horizon $T = 10\,000$ steps. As such, all numbers in Table 1 are means of means.

Each algorithms have several *hyper-parameters* which are algorithmic parameters that are tweaked before the simulation begins. They were specified with a meta-model in COMBO and optimized with a random forest bandit. The following summary contain the most important hyper-parameters of the respective algorithms:

Random Baseline for comparison. The configurations are generated uniformly at random with the only criteria that they should satisfy the constraints.

Oracle Baseline for comparison. This uses the local search optimizer to maximize the surrogate user model directly, which the other bandits do not have direct access to. The global maximum for auto complete is 0.3953 and for top- k categories it is 0.1031. The numbers for Oracle in Table 1 show how far from optimal the local search optimizer is on average. Since the surrogate user model has additional interaction terms the optimization problem is harder for Oracle than for some algorithms.

Table 1: Simulation results of bandit optimization algorithms evaluated on both feature cases: auto complete search (AC) and top- k categories for $k = 5$. *Mean rewards* is the maximization target. *Choose* and *update* is how long time the algorithm takes to choose a configuration and update it, respectively.

Algorithm	Choose (ms)		Update (ms)		Mean rewards (SD)	
	AC	Top-k	AC	Top-k	AC	Top-k
Random	0.10	0.81	-	-	0.0136 (9e-4)	0.0879 (1e-5)
Oracle	3.91	81.86	-	-	0.3658 (9e-4)	0.0970 (9e-5)
DT	0.16	2.10	0.03	0.03	0.2131 (0.05)	0.0888 (4e-4)
RF	0.99	43.58	0.13	2.61	0.2501 (0.04)	0.0895 (8e-4)
GLM _{diag}	1.52	25.12	0.01	0.01	0.1635 (0.05)	0.0891 (2e-5)
GLM _{full}	1.64	25.35	0.10	8.39	0.2335 (0.04)	0.0891 (2e-5)
NL	8.08	32.68	7.41	12.54	0.0435 (0.06)	0.0881 (6e-2)

DT Decision tree bandit. The hyper-parameters were related to how significant a split should be (δ and τ -parameter of VFDT [84]).

RF Random forest bandit with 200 decision tree bandits (as in the DT above). The number of trees in the forest does have an impact on performance but has diminishing payoff after 100. The hyper-parameters were both the parameters from the decision tree bandit and parameters for the bootstrapping procedure of standard random forest, i.e., how many variables and data points each tree should have.

GLM_{diag} Generalized linear model bandit with a simplified diagonalized covariance vector. The hyper-parameters were the prior on the weights, regularization factor, and an exploration factor.

GLM_{full} The same as the above GLM_{diag} but with a full covariance matrix. The hyper-parameters were the also same.

NL Neural linear bandit with logit output, ReLU activation on the hidden layers, and weight-decay regularization. The neural network optimizer was RMSProp running on a CPU. Since neural linear combines the representation power of a neural network with a GLM_{full} bandit, it inherits the hyper-parameters of the GLM bandit. It also has hyper-parameters in the number of hidden layers and their widths, mini-batch size of updates to the network, learning rate of the optimizer, number of epochs to repeat data points, and initialization noise of the weights.

Table 1 summarizes the results of all algorithms for the feature cases, with both mean calculation time and mean rewards. The dimensions are as follows. *Mean rewards* is the maximization target and is the average expected reward per repetition. The numbers in parentheses are standard deviations between repetition runs for mean rewards. Algorithms with high standard deviation tend to get stuck in local optima for some runs of the simula-

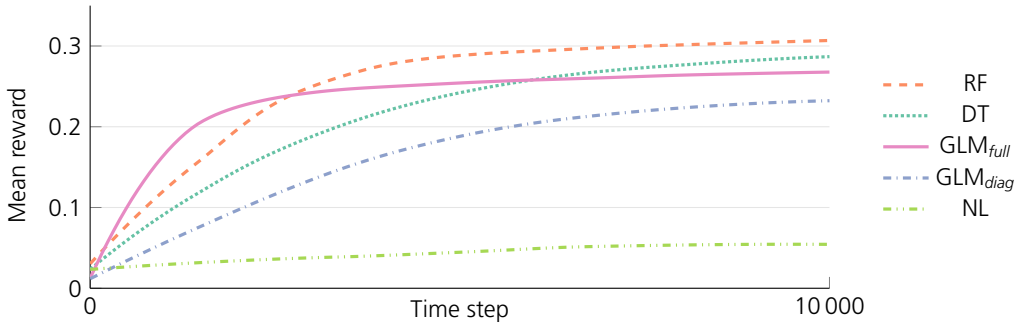


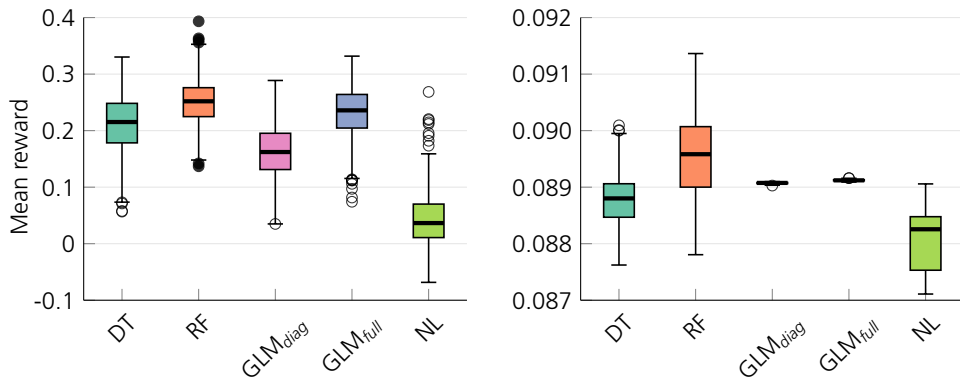
Figure 8: Mean rewards per repetition over time on the auto complete feature for the bandit optimization algorithms. The maximization target is the area under the curve.

tions. *Choose* measures how long the bandit algorithm takes to generate a software configuration in milliseconds and *update* time how long it takes to update the bandit optimization algorithm. The choose time is critical to keep low in order to not degrade user experience, the update times are not as critical as long as they are not too excessive since they cannot be fully parallelized.

The figures 8 and 9 further illustrate the differences in performance of the algorithms. Figure 8 shows the performance over each time step averaged over the simulation repetitions. The figure illustrates that the specific choice of time horizon to $T = 10\,000$ does have an impact on performance, if the simulation would continue for 10 or 100 times as long the ranking of the algorithms might very well have been different, i.e. it might be the case that the neural linear (NL) bandit eventually improves. Figure 9 show the overall variation in mean rewards between repetitions with boxplots on the quartiles and outliers outside the quartile range.

The results show that the random forest (RF) bandit perform well in terms of rewards for both auto complete and top-k and the neural linear (NL) bandit has poor performance. Clearly, the NL bandit needs more data to converge. As evident from Figure 9, the performance difference between the algorithms is more pronounced in the top- k categories feature. Here it is clear that the random forest bandit is needed for its improved representational power in comparison to the simpler models. However, as can be seen in Figure 8, the generalized linear model bandit with full covariance matrix (GLM_{full}) converges quicker in the auto complete feature.

It should be noted that some algorithms are favored over others in the simulation due to the choices in the surrogate user model. That is, the machine learning model in the top- k categories' surrogate user model is the same as the one used by the neural linear (NL) bandit. Also, we did not add any interaction terms to the linear bandits, which could have



(a) Auto complete feature.

(b) Top- k categories feature.

Figure 9: Mean rewards obtained per repetition in the simulations for each bandit optimization algorithm.

been done. We reasoned that there will almost always be unmodeled interaction terms in real world applications. Since the surrogate user model for auto complete had pairwise interaction terms only, the linear models would have no interaction terms. In the top- k categories feature the performance would probably improve as well with interaction terms.

Initially, before adding hyper-parameter search, the GLM and NL bandits performed much worse than their final performance. Thus, they derived higher benefit from tuned hyper-parameters. As such, the applicability for GLM and NL are dependent on having a simulation environment in which to search for hyper-parameters. In addition, NL is very hard to tune correctly in comparison to the other methods, as evident from the wide performance spread and the large number of hyper-parameters. The outcome of network architecture is also hard to predict. Adding more network nodes or layers increases the representational power of the algorithm, but increases convergence time which also affects performance.

Regarding the time estimates in Table 1, all bandit algorithms are usable within reasonable time; with a choose time below 50ms. The clear winner in both choose and update times is the decision tree (DT) bandit algorithm. The effect of scaling to more variables on choose and update times can also be seen in the table. The choose time for the RF bandit (see Section 5.2.1) scales poorly, though it can be controlled by limiting the number of trees and the number of nodes and variables per tree. The simulations for the neural linear (NL) bandit are slowest since they are dominated by the non-parallelizable updates.

All the bandit optimization algorithms presented here have trade-offs between performance and choose and update time efficiency. For example, the maximum number of nodes in decision tree (DT) bandit, the number of trees in random forest (RF) bandit, the number of hidden layers in neural linear (NL) bandit, or the number of interaction terms in the

generalized linear model (GLM) bandit. We have not fully explored this trade-off, other than stating the specific choices we made. We suggest that when deciding on an actual algorithm, developers should start with the threshold on choose time that is acceptable and then find the best algorithm that stays below the threshold.

In summary, the random forest (RF) bandit is the clear winner in the simulations and should serve well as a good default choice. It achieves highest performance for both feature cases and had few outliers with runs of poor performance (c.f., Figure 9).

7 Discussion

We introduced the Constraint Oriented Multi-variate Bandit Optimization (COMBO) toolkit, used for designing software that improves over time. Using the toolkit, each user receives its own configuration and the toolkit optimizes the overall user experience or business value. It contains machine learning algorithms and constraint solvers, that can be applied to optimize a search space that is specified in a domain specific language. We used the toolkit to model the variability of two features in collaboration with an e-commerce company called Apptus. Thereby we demonstrated that the toolkit can be applied in industry relevant settings. In this section, the implications of when this is put into practice is discussed.

7.1 From continuous experimentation to continuous optimization

We define *continuous optimization* as a practice that extends continuous experimentation further by having an algorithm optimize software for better product user experience and business value. The practice entails having a decision algorithm jointly co-optimize the variables in a production environment with user data. As with continuous experimentation, the variables in the optimization can be anything, e.g., details in the visual design and layout, user experience flow, or algorithmic parameters on a web server that impact user experience.

We see two main reasons for that continuous optimization will improve products. First, very large search spaces can be explored by an algorithm. This means that the optimization algorithm might find solutions that developers might otherwise overlook. Mikkulainen et al. [208] mention this as a common occurrence in their commercial tool for visual design. Second, according to Hill et al. [138], it enables personalization of software to users by having variables that describe users in the optimization (e.g., device type and location). As many parameters as needed can be added to the search space in order to finely segment users so that the algorithm can find different solutions for different users.

According to Fitzgerald et al. [113], discontinuous development is more important than continuous, meaning that product innovation is more important than refinement. We believe that the introduction of a toolkit like COMBO to a development process is not a contradiction to this. Following the reasoning by Mikkulainen et al. [208], continuous optimization can de-emphasize narrow incrementalism by offloading parts of the decision making process so that developers can focus on the more important parts. Thus, continuous optimization offers a complementary approach to software design, by loosely specifying implementation details and letting the algorithm decide instead.

Based on the procedure used to build the feature cases in Section 6, we propose a process for how continuous optimization should be conducted in industry with continuous software development. The validation of the process is preliminary. The four steps of the process are: (1) investigate and prioritize the variability of the feature by user experience (UX) research methods, data mining, or prototype experiments, (2) formulate a model of the variables in the optimization search space and add constraints to prune invalid configurations, (3) tweak the algorithmic performance in offline simulations, and finally (4) validate the solution in a controlled experiment with real users. The process can then restart from step 1.

This process is similar to one that is reportedly used for developing recommender systems at Netflix [5] and Apttus [247]. Simulations are also used at both companies to evaluate changes to their recommender system in fast feedback cycles. If the effect of a change is evaluated to be positive in the simulation, then the change is deployed and subjected to a controlled environment (i.e. an A/B test) in a production environment on real users.

7.2 Considerations for what metric and changes to optimize for

Experimentation and optimization is done with respect to a given metric. Though, quantifying business value in metrics is a well documented challenge for many software companies [104, 188, 224, 320]. Having many metrics in an experiment is one coping strategy, in the hope that together they point in the right direction. Hundreds of metrics are reportedly used for a single experiment at Microsoft [164, 193]. For optimization, multi-objective optimization [215, 283] can be applied offline at compile time, where someone can manually make a trade-off between metrics from a Pareto front. However, for the online bandit optimization algorithms, a single metric is required to serve as rewards (though that metric can be a scalar index).

As mentioned in Section 4.1, there are established metrics in e-commerce that measure business value. Optimizing for revenue or profit is possible but only few users convert to paying customers. Updates to the algorithm will also be delayed due to having to wait until a session expires to determine that they did not convert. In addition, e-commerce companies will have different business models that can result in needing other metrics.

For example, they might want to push a certain product since they are overstocked or they might want as many consumers as possible to sign up to their loyalty club to have a recurring source of revenue. All of these volatile factors make optimizing for business value challenging.

In e-commerce, it is unlikely that a change in the user interface will instill a purchasing need in consumers. The way that an optimization algorithm can affect business value is rather by removing hurdles in the user experience that would otherwise make a consumer turn to a competitor. For example, a web shop can be judged to be untrustworthy due to the impression it gives from its design, then customers would not want to buy from there. Consequently, it might be better to optimize with user experience metrics, since all users can contribute data to a user experience metric even if they do not convert to paying customers. Therefore we argue that, under these circumstances, user experience metrics should be the default choice for algorithmic optimization before business value metrics.

User experience metrics come with their own set of challenges. For instance, optimizing the number of clicks on one area of the user interface will probably lead to a local improvement, but possibly at the expense of other areas of the site. This effect where changes shift clicks around between areas is known as *cannibalization* [81, 175]. If the area that gets cannibalized has higher business value than the cannibalizing area the optimization can even be detrimental. This has been researched at Apptus [40] for their recommender system in particular. If the recommender system is optimized for clicks, it can distract consumers with interesting products, rather than products that they might buy.

Ultimately, different parts of the user interface can be optimized for different metrics. Each interface component is designed to fulfill a goal and that is the goal to be quantified. Returning to the feature cases from Section 6, in the auto complete feature the goal was to aid users in the discovery of products through the search. Thus, whether they used the aid or not—measured in click-through-rate—is a reasonably risk free optimization metric. In the top- k categories, using clicks seems riskier since some categories can distract users or might lead them to believe that the store does not sell certain products that they do sell. For that reason, business value metrics seem fitting, even though it will lead to a slower optimization due to less available data. Whether the reasoning behind choosing a given metric is correct or not is something that can be validated in an A/B test once the optimization system is put into production.

7.3 Future directions

Future work is required to strengthen the evaluation of COMBO to make a stronger claim about the applicability and suitability of the toolkit. A more thorough continuous optimization process would be useful as well. Below we present remaining technical barriers for wide-spread adoption of tools like COMBO that require further study, in no particular order.

7.3.1 Ramifications on software quality and testing

The continuous optimization practice that we advocate for is not without risks. Both the need to maintain machine learning models and the increased variability of software, caused by optimization, are challenging to handle on their own. Sculley et al. [269] from Google describe how maintaining machine learning models is a source of technical debt, this has also been studied extensively by software quality researchers [195]. Regarding software testing, the model-based approach [55, 159] to experimentation [48] and optimization can be used both for formal verification and software testing (see Section 5.1). Much has been published on model-based testing [289], also for user interfaces specifically [271]. Since a model is built regardless for the optimization problem, focusing more on integrating support for the model-based testing techniques in the toolkit would be fruitful.

7.3.2 Concept drift for bandit optimization

In machine learning, concept drift occurs when the environment that the machine learning model has been trained in changes over time, i.e., if users change their behavior. There are general approaches to detect and be more robust against concept drift, such as Minku and Yao [210], and specific solutions to software engineering related applications [160, 182]. Those approaches cannot be directly applied to this work since detection of concept drift is not enough. For univariate multi-armed bandits, the solution is usually to apply a moving window to the arms' descriptive statistics [40, 47]. This adds more complexity to the solution since another hyper-parameter needs to be estimated, i.e., how long the window should be. Also, unlearning specific data points for multi-variate multi-armed bandit is harder than unlearning for descriptive statistics, due to the learned machine learning model. If possible, the approach should avoid complete re-training since that implies needing to store user data permanently. Thus, more work is required to study the impact and approaches for concept drift in this domain.

7.3.3 Continuous model updates

Adding or removing variables to the optimization search space must be done without restarting the optimization. For generalized linear models, decision trees, and random forests, model updates are straightforward to both implement and make inference about. For neural networks the effect of a change will be unpredictable, the algorithm might not even converge to a new solution so it could be better to restart the model. This is further discussed in the technical debt for machine learning paper [269]. Furthermore, for all bandit algorithms, users will have their existing configurations be invalidated when the underlying model is updated. Their configuration in the new model's search space should preferably not be drastically different from their old one to avoid user confusion. We plan to add support to the toolkit for generating configurations that minimize the distance between a configuration of an old search space to a new one; while at the same time maximizing the expected reward of the new configuration.

7.3.4 Bandit optimization algorithms

Several bandit optimization algorithms are included in the COMBO toolkit. Still, we have only begun to explore all the design options for algorithms, especially for neural networks and deep learning. Additionally, ensembles of several bandit optimization algorithms could be applied to improve performance by initially using algorithms that learn quickly and then gradually switching to slower algorithms with better representational power that eventually outperform simpler methods.

7.3.5 Algorithm tuning and cold start

Based on the simulation evaluation in Section 6.3 we advise against just using default settings on a bandit optimization algorithm. The performance can improve drastically by tuning hyper-parameters in simulations, though we refrain from giving numbers on the improvement since the default parameters are somewhat arbitrarily set in the first place. However, this will be hard when implementing a new feature with no data. This was the case for the auto complete feature in Section 6.1 where data was manually collected. This might be too expensive for regular software development. Regardless, the constraint oriented approach in the COMBO toolkit can be used here to exclude incompatible design choices observed during feature development (possibly in conjunction with informative Bayesian priors). The situation is analogous to the *cold start* problem in recommender systems [261]. That occurs when new products are added that do not have any associated behavioral data. Thus, connections could be drawn to this research field.

8 Conclusions

In e-commerce, continuous experimentation is widespread to optimize web shops; high-profile companies have recently adopted online machine learning based optimization methods to both increase scale of the optimizations, and to have personalized software to different user's needs. This technology is readily available but can be hard to implement on anything other than superficial details in the user interface. In this work, the open-source toolkit COMBO is introduced to support the algorithmic optimization at a validation company in e-commerce. The toolkit can be used for building data-driven software features that learn from user behavior in terms of user experience or business value. We have shown that modeling software hierarchies in a formal model enables algorithmic optimization of complex software. Thus, the toolkit extends optimization to more use cases. There are still many further opportunities for future work in this domain to enable adoption in other fields than e-commerce. However, we insist that the toolkit can be used today to improve the user experience and business value of software.

Acknowledgements

Thanks to Per Runeson, Elizabeth Bjarnason, Luigi Nardi, and the three anonymous reviewers for providing useful feedback to this manuscript.

This work was partially supported by the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation.

Appendix

A Replication of simulations

To replicate the simulations in Section 6.3, run the following commands on a Unix terminal with a JDK and git:

```
$ git clone https://github.com/rasros/combo.git
$ cd combo
$ ./gradlew assemble
$ # runs the random forest bandit on the auto complete data set:
$ ./jvm-demo/simulation.sh RF AC
$ # view a list of options, algorithms, and data sets:
$ ./jvm-demo/simulation.sh -h
```

References

- [1] Gabriel Abend. The meaning of ‘theory’. *Sociological Theory*, 26(2):173–199, 2008. doi:[10.1111/j.1467-9558.2008.00324.x](https://doi.org/10.1111/j.1467-9558.2008.00324.x).
- [2] Vineet Abhishek and Shie Mannor. A nonparametric sequential test for online randomized experiments. In *Proceedings of the 26th International Conference on World Wide Web Companion*, WWW, pages 610–616, 2017. doi:[10.1145/3041021.3054196](https://doi.org/10.1145/3041021.3054196).
- [3] Muhammad Adinata and Inggriani Liem. A/B test tools of native mobile application. In *Proceedings of the International Conference on Data and Software Engineering*, ICODSE, pages 1–6, 2014. doi:[10.1109/icodse.2014.7062683](https://doi.org/10.1109/icodse.2014.7062683).
- [4] Steve Adolph, Wendy Hall, and Philippe Kruchten. Using grounded theory to study the experience of software development. *Empirical Software Engineering*, 16(4):487–513, 2011. doi:[10.1007/s10664-010-9152-6](https://doi.org/10.1007/s10664-010-9152-6).
- [5] Xavier Amatriain. Beyond data: From user information to business value through personalized recommendations and consumer science. In *Proceedings of the 22nd International Conference on Information and Knowledge Management*, CIKM, pages 2201–2208, 2013. doi:[10.1145/2505515.2514701](https://doi.org/10.1145/2505515.2514701).
- [6] Marc Andreessen. Part 4: The only thing that matters. Accessed: 2021-12-10, 2007. pmarchive.com/guide_to_startups_part4.html.
- [7] Nirupama Appikala, Miao Chen, Michael Natkovich, and Joshua Walters. Demystifying dark matter for online experimentation. In *Proceedings of the 5th International Conference on Big Data*, BigData, pages 1620–1626, 2017. doi:[10.1109/bigdata.2017.8258096](https://doi.org/10.1109/bigdata.2017.8258096).
- [8] Andrea Arcuri and Gordon Fraser. On parameter tuning in search based software engineering. In *Proceedings of the 3rd International Symposium on Search Based Software Engineering*, SSBSE, pages 33–47, 2011. doi:[10.1007/978-3-642-23716-4_6](https://doi.org/10.1007/978-3-642-23716-4_6).

- [9] Florian Auer and Michael Felderer. Current state of research on continuous experimentation: A systematic mapping study. In *Proceedings of the 44th Euromicro Conference on Software Engineering and Advanced Applications*, SEAA, pages 335–344, 2018. doi:[10.1109/SEAA.2018.00062](https://doi.org/10.1109/SEAA.2018.00062).
- [10] Florian Auer and Michael Felderer. An infrastructure for platform-independent experimentation of software changes. In *Proceedings of the 47th International Conference on Current Trends in Theory and Practice of Informatics*, SOFSEM, pages 445–457, 2021. doi:[10.1007/978-3-030-65854-0_11](https://doi.org/10.1007/978-3-030-65854-0_11).
- [11] Florian Auer, Rasmus Ros, Lukas Kaltenbrunner, Per Runeson, and Michael Felderer. Controlled experimentation in continuous experimentation: Knowledge and challenges. *Information and Software Technology*, 134:106551, 2021. doi:[10.1016/j.infsof.2021.106551](https://doi.org/10.1016/j.infsof.2021.106551).
- [12] Florian Auer, Rasmus Ros, Lukas Kaltenbrunner, Per Runeson, and Michael Felderer. Dataset of controlled experimentation in continuous experimentation: Knowledge and challenges. *figshare*, 2021. doi:[10.6084/m9.figshare.13712329](https://doi.org/10.6084/m9.figshare.13712329).
- [13] Francisco Galuppo Azevedo, Bruno Demattos Nogueira, Fabricio Murai, and Ana Paula C. Silva. Estimation errors in network A/B testing due to sample variance and model misspecification. In *Proceedings of the 8th International Conference on Web Intelligence*, WI, pages 540–545, 2018. doi:[10.1109/wi.2018.00-40](https://doi.org/10.1109/wi.2018.00-40).
- [14] Lars Backstrom and Jon Kleinberg. Network bucket testing. In *Proceedings of the 20th international conference on World Wide Web*, WWW, pages 615–624, 2011. doi:[10.1145/1963405.1963492](https://doi.org/10.1145/1963405.1963492).
- [15] E. Bakshy, D. Eckles, and M. S. Bernstein. Designing and deploying online field experiments. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW, pages 283–292, 2014. doi:[10.1145/2566486.2567967](https://doi.org/10.1145/2566486.2567967).
- [16] Eytan Bakshy and Dean Eckles. Uncertainty in online experiments with dependent data: An evaluation of bootstrap methods. In *Proceedings of the 19th International Conference on Knowledge Discovery and Data Mining*, KDD, pages 1303–1311, 2013. doi:[10.1145/2487575.2488218](https://doi.org/10.1145/2487575.2488218).
- [17] Rajiv D. Banker, Srikant M. Datar, Chris F. Kemerer, and Dani Zweig. Software complexity and maintenance costs. *Communications of the ACM*, 36(11):81–95, 1993. doi:[10.1145/163359.163375](https://doi.org/10.1145/163359.163375).
- [18] Blake Bartlett. What is product led growth? how to build a software company in the end user era. Accessed: 2021-01-15, 2020. openviewpartners.com/blog/what-is-product-led-growth.

- [19] Victor R. Basili. Quantitative evaluation of software methodology. In *Proceedings of the 1st Pan Pacific Computer Conference*, volume 1, pages 379–398, 1985.
- [20] Kent Beck. Embracing change with extreme programming. *Computer*, 32(10):70–77, 1999. doi:[10.1109/2.796139](https://doi.org/10.1109/2.796139).
- [21] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6):615–636, 2010. doi:[10.1016/j.is.2010.01.001](https://doi.org/10.1016/j.is.2010.01.001).
- [22] Raquel Benbunan-Fich. The ethics of online research with unsuspecting users: From A/B testing to C/D experimentation. *Research Ethics*, 13(3-4):200–218, 2017. doi:[10.1177/1747016116680664](https://doi.org/10.1177/1747016116680664).
- [23] Daniel Le Berre and Anne Parrain. The SAT4J library, release 2.2, system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(2–3):59–64, 2010. doi:[10.3233/SAT190075](https://doi.org/10.3233/SAT190075).
- [24] Danilo Beuche, Holger Papajewski, and Wolfgang Schröder-Preikschat. Variability management with feature models. *Science of Computer Programming*, 53(3):333–352, 2004. doi:[10.1016/j.scico.2003.04.005](https://doi.org/10.1016/j.scico.2003.04.005).
- [25] Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of satisfiability*. IOS Press, 2009.
- [26] Elizabeth Bjarnason. Prototyping practices in software startups: Initial case study results. In *Proceedings of the 29th International Requirements Engineering Conference Workshops*, REW, pages 206–211, 2021. doi:[10.1109/REW53955.2021.00038](https://doi.org/10.1109/REW53955.2021.00038).
- [27] Jens Björklund, Jens Ljungblad, and Jan Bosch. Lean product development in early stage startups. In *Proceedings of the 1st International Workshop on From Start-ups to SaaS Conglomerate: Life Cycles of Software Products Workshop*, IW-LCSP, pages 19–32, 2013.
- [28] Grady Booch. *Object oriented design: with applications*. Benjamin-Cummings, 1991.
- [29] Markus Borg. TuneR: a framework for tuning software engineering tools with hands-on instructions in R. *Journal of Software: Evolution and Process*, 28(6):427–459, 2016. doi:[10.1002/smr.1784](https://doi.org/10.1002/smr.1784).
- [30] Slava Borodovsky and Saharon Rosset. A/B testing at SweetIM: The importance of proper statistical analysis. In *Proceedings of the 11th International Conference on Data Mining Workshops*, ICDMW, pages 733–740, 2011. doi:[10.1109/ICDMW.2011.19](https://doi.org/10.1109/ICDMW.2011.19).

- [31] Jan Bosch. Building products as innovation experiment systems. In *Proceedings of the 3rd International Conference on Software Business*, ICSOB, pages 27–39, 2012. doi:[10.1007/978-3-642-30746-1_3](https://doi.org/10.1007/978-3-642-30746-1_3).
- [32] Jan Bosch. Speed, data, and ecosystems: The future of software engineering. *IEEE Software*, 33(1):82–88, 2016. doi:[10.1109/MS.2016.14](https://doi.org/10.1109/MS.2016.14).
- [33] Jan Bosch and Ulrik Eklund. Eternal embedded software: Towards innovation experiment systems. In *Proceedings of the 5th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*, ISoLA, pages 19–31, 2012. doi:[10.1007/978-3-642-34026-0_3](https://doi.org/10.1007/978-3-642-34026-0_3).
- [34] Jan Bosch and Helena Holmström Olsson. Data-driven continuous evolution of smart systems. In *Proceedings of the 11th International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS, pages 28–34, 2016. doi:[10.1145/2897053.2897066](https://doi.org/10.1145/2897053.2897066).
- [35] Jan Bosch, Helena Holmström Olsson, Jens Björk, and Jens Ljungblad. The early stage software startup development model: a framework for operationalizing lean principles in software startups. In Brian Fitzgerald, Kieran Conboy, Ken Power, Ricardo Valerdi, Lorraine Morgan, and Klaas-Jan Stol, editors, *Lean Enterprise Software and Systems*, pages 1–15. Springer Publishing Company, 2013. doi:[10.1007/978-3-642-44930-7_1](https://doi.org/10.1007/978-3-642-44930-7_1).
- [36] Jan Bosch, Helena Holmström Olsson, and Ivica Crnkovic. It takes three to tango: Requirement, outcome/data, and AI driven development. In *Proceedings of the 1st International Workshop on Software-intensive Business: Start-ups, Ecosystems and Platforms*, SiBW, pages 177–192, 2018.
- [37] Petra Bosch-Sijtsema and Jan Bosch. User involvement throughout the innovation process in high-tech industries. *Journal of Product Innovation Management*, 32(5): 793–807, 2015. doi:[10.1111/jpim.12233](https://doi.org/10.1111/jpim.12233).
- [38] Douglas Bowman. Goodbye, Google. Accessed: 2022-01-24, 2009. stopdesign.com/archive/2009/03/20/goodbye-google.html.
- [39] Manuel Brhel, Hendrik Meth, Alexander Maedche, and Karl Werder. Exploring principles of user-centered agile software development: A literature review. *Information and Software Technology*, 61:163–181, 2015. doi:[10.1016/j.infsof.2015.01.004](https://doi.org/10.1016/j.infsof.2015.01.004).
- [40] Björn Brodén, Mikael Hammar, Bengt J. Nilsson, and Dimitris Paraschakis. Bandit algorithms for e-Commerce recommender systems. In *Proceedings of the 11th Conference on Recommender Systems*, pages 349–349, 2017. doi:[10.1145/3109859.3109930](https://doi.org/10.1145/3109859.3109930).

- [41] Björn Brodén, Mikael Hammar, Bengt J. Nilsson, and Dimitris Paraschakis. A bandit-based ensemble framework for exploration/exploitation of diverse recommendation components: An experimental study within e-commerce. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 10(1):1–32, 2019. doi:[10.1145/3237187](https://doi.org/10.1145/3237187).
- [42] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012. doi:[10.1109/TCIAIG.2012.2186810](https://doi.org/10.1109/TCIAIG.2012.2186810).
- [43] Sabine Brunswicker, Cara Wrigley, and Sam Bucolo. Business model experimentation: What is the role of design-led prototyping in developing novel business models? In Martin Curley and Piero Formica, editors, *The experimental nature of new venture creation: capitalizing on Open Innovation 2.0*, pages 139–151. Springer Publishing Company, 2013. doi:[10.1007/978-3-319-00179-1_13](https://doi.org/10.1007/978-3-319-00179-1_13).
- [44] Tomasz Buchert, Cristian Ruiz, Lucas Nussbaum, and Olivier Richard. A survey of general-purpose experiment management tools for distributed systems. *Future Generation Computer Systems*, 45:1–12, 2015. doi:[10.1016/j.future.2014.10.007](https://doi.org/10.1016/j.future.2014.10.007).
- [45] Roman Budylin, Alexey Drutsa, Ilya Katsev, and Valeriya Tsoy. Consistent transformation of ratio metrics for efficient online controlled experiments. In *Proceedings of the 11th International Conference on Web Search and Data Mining*, WSDM, pages 55–63, 2018. doi:[10.1145/3159652.3159699](https://doi.org/10.1145/3159652.3159699).
- [46] Mario Bunge. *Philosophy of science: volume 2, from explanation to justification*. Routledge, 1998.
- [47] Giuseppe Burtini, Jason Loepky, and Ramon Lawrence. A survey of online experiment design with the stochastic multi-armed bandit. *arXiv:1510.00757v4*, [stat.ML], 2015.
- [48] Javier Cámara and Alfred Kobsa. Facilitating controlled tests of website design changes: A systematic approach. In *Proceedings of the 9th International Conference on Web Engineering*, ICWE, pages 370–378, 2009. doi:[10.1007/978-3-642-02818-2_30](https://doi.org/10.1007/978-3-642-02818-2_30).
- [49] Rocío Cañamares, Marcos Redondo, and Pablo Castells. Multi-armed recommender system bandit ensembles. In *Proceedings of the 13th Conference on Recommender Systems*, RecSys, pages 432–436, 2019. doi:[10.1145/3298689.3346984](https://doi.org/10.1145/3298689.3346984).
- [50] Deepayan Chakrabarti, Ravi Kumar, Filip Radlinski, and Eli Upfal. Mortal multi-armed bandits. In *Proceedings of the 21st International Conference on Neural Information Processing Systems*, NIPS, pages 273–280, 2008.

- [51] Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. A scalable and nearly uniform generator of SAT witnesses. In *Proceedings of the 25th International Conference on Computer Aided Verification, CAV*, pages 608–623, 2013. doi:[10.1007/978-3-642-39799-8_40](https://doi.org/10.1007/978-3-642-39799-8_40).
- [52] Stephanie Chamberlain, Helen Sharp, and Neil Maiden. Towards a framework for integrating agile development and user-centred design. In *Proceedings of the 7th International Conference on Extreme Programming and Agile Processes in Software Engineering, XP*, pages 143–153, 2006. doi:[10.1007/11774129_15](https://doi.org/10.1007/11774129_15).
- [53] Ximena Alejandra Flechas Chaparro and Leonardo Augusto de Vasconcelos Gomes. Pivot decisions in startups: a systematic literature review. *International Journal of Entrepreneurial Behavior & Research*, 27(4), 2021. doi:[10.1108/IJEBR-12-2019-0699](https://doi.org/10.1108/IJEBR-12-2019-0699).
- [54] Olivier Chapelle and Lihong Li. An empirical evaluation of thompson sampling. In *Proceedings of the 24th International Conference on Neural Information Processing Systems, NIPS*, pages 2249–2257, 2011.
- [55] Lianping Chen, Muhammad Ali Babar, and Nour Ali. Variability management in software product lines: a systematic review. In *Proceedings of the 13th International Software Product Line Conference, SPLC*, pages 81–90, 2009.
- [56] Nanyu Chen, Min Liu, and Ya Xu. How A/B tests could go wrong: Automatic diagnosis of invalid online experiments. In *Proceedings of the 12th International Conference on Web Search and Data Mining, WSDM*, pages 501–509, 2019. doi:[10.1145/3289600.3291000](https://doi.org/10.1145/3289600.3291000).
- [57] Russell Chen, Miao Chen, Mahendrasinh Ramsinh Jadav, Joonsuk Bae, and Don Matheson. Faster online experimentation by eliminating traditional A/A validation. In *Proceedings of the 5th International Conference on Big Data, BigData*, pages 1635–1641, 2017. doi:[10.1109/bigdata.2017.8258098](https://doi.org/10.1109/bigdata.2017.8258098).
- [58] Wei Chen, Yajun Wang, and Yang Yuan. Combinatorial multi-armed bandit: General framework and applications. In *Proceedings of the 30th International Conference on Machine Learning, PMLR*, pages 151–159, 2013.
- [59] Henry Chesbrough. Business model innovation: it’s not just about technology anymore. *Strategy & Leadership*, 35(6):12–17, 2007. doi:[10.1108/10878570710833714](https://doi.org/10.1108/10878570710833714).
- [60] Henry Chesbrough and Richard S. Rosenbloom. The role of the business model in capturing value from innovation: evidence from Xerox Corporation’s technology spin-off companies. *Industrial and Corporate Change*, 11(3):529–555, 2002. doi:[10.1093/icc/11.3.529](https://doi.org/10.1093/icc/11.3.529).

- [61] David Choi. Estimation of monotone treatment effects in network experiments. *Journal of the American Statistical Association*, 112(519):1147–1155, 2017. doi:[10.1080/01621459.2016.1194845](https://doi.org/10.1080/01621459.2016.1194845).
- [62] Emmanuelle Claeys, Pierre Gançarski, Myriam Maumy-Bertrand, and Hubert Wassner. Regression tree for bandits models in A/B testing. In *Proceedings of the 16th International Symposium on Advances in Intelligent Data Analysis, IDA*, pages 52–62, 2017. doi:[10.1007/978-3-319-68765-0_5](https://doi.org/10.1007/978-3-319-68765-0_5).
- [63] Paul Clarke and Rory V. O’Connor. The situational factors that affect the software development process: Towards a comprehensive reference framework. *Information and Software Technology*, 54(5):433–447, 2012. doi:[10.1016/j.infsof.2011.12.003](https://doi.org/10.1016/j.infsof.2011.12.003).
- [64] Dominic Coey and Michael Bailey. People and cookies: Imperfect treatment assignment in online experiments. In *Proceedings of the 25th International Conference on World Wide Web, WWW*, pages 1103–1111, 2016. doi:[10.1145/2872427.2882984](https://doi.org/10.1145/2872427.2882984).
- [65] European Commission. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (General Data Protection Regulation), 2016. eli:[reg/2016/679/2016-05-04](https://eur-lex.europa.eu/eli/reg/2016/679/2016-05-04).
- [66] Mauro Conti, Ankit Gangwal, Sarada Prasad Gochhayat, and Gabriele Tolomei. Spot the difference: Your bucket is leaking: A novel methodology to expose A/B testing effortlessly. In *Proceedings of the 6th Conference on Communications and Network Security, CNS*, pages 1–7, 2018. doi:[10.1109/cns.2018.8433122](https://doi.org/10.1109/cns.2018.8433122).
- [67] Thomas Crook, Brian Frasca, Ron Kohavi, and Roger Longbotham. Seven pitfalls to avoid when running controlled experiments on the web. In *Proceedings of the 15th International Conference on Knowledge Discovery and Data Mining, KDD*, pages 1105–1114, 2009. doi:[10.1145/1557019.1557139](https://doi.org/10.1145/1557019.1557139).
- [68] Daniela S. Cruzes and Tore Dybå. Recommended steps for thematic synthesis in software engineering. In *Proceedings of the 5th International Symposium on Empirical Software Engineering and Measurement, ESEM*, pages 275–284, 2011. doi:[10.1109/ESEM.2011.36](https://doi.org/10.1109/ESEM.2011.36).
- [69] Daniela S. Cruzes, Tore Dybå, Per Runeson, and Martin Höst. Case studies synthesis: A thematic, cross-case, and narrative synthesis worked example. *Empirical Software Engineering*, 20(6):1634–1665, 2015. doi:[10.1007/s10664-014-9326-8](https://doi.org/10.1007/s10664-014-9326-8).
- [70] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Staged configuration using feature models. In Robert L. Nord, editor, *Software Product Lines*, pages 266–283. Springer Publishing Company, 2004. doi:[10.1007/978-3-540-28630-1_17](https://doi.org/10.1007/978-3-540-28630-1_17).

- [71] Ariyam Das and Harish Ranganath. When web personalization misleads bucket testing. In *Proceedings of the 1st Workshop on User Engagement Optimization*, UEO, pages 17–20, 2013. doi:[10.1145/2512875.2512879](https://doi.org/10.1145/2512875.2512879).
- [72] George S. Day. The capabilities of market-driven organizations. *Journal of Marketing*, 58(4):37–52, 1994. doi:[10.1177/002224299405800404](https://doi.org/10.1177/002224299405800404).
- [73] William Edwards Deming. *Out of the Crisis*. MIT Press, 1986.
- [74] Alex Deng. Objective bayesian two sample hypothesis testing for online controlled experiments. In *Proceedings of the 24th International Conference on World Wide Web*, WWW, pages 923–928, 2015. doi:[10.1145/2740908.2742563](https://doi.org/10.1145/2740908.2742563).
- [75] Alex Deng and Victor Hu. Diluted treatment effect estimation for trigger analysis in online controlled experiments. In *Proceedings of the 8th International Conference on Web Search and Data Mining*, WSDM, pages 349–358, 2015. doi:[10.1145/2684822.2685307](https://doi.org/10.1145/2684822.2685307).
- [76] Alex Deng and Xiaolin Shi. Data-driven metric development for online controlled experiments: Seven lessons learned. In *Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining*, KDD, pages 77–86, 2016. doi:[10.1145/2939672.2939700](https://doi.org/10.1145/2939672.2939700).
- [77] Alex Deng, Ya Xu, Ron Kohavi, and Toby Walker. Improving the sensitivity of online controlled experiments by utilizing pre-experiment data. In *Proceedings of the 6th International Conference on Web Search and Data Mining*, WSDM, pages 123–132, 2013. doi:[10.1145/2433396.2433413](https://doi.org/10.1145/2433396.2433413).
- [78] Alex Deng, Tianxi Li, and Yu Guo. Statistical inference in two-stage online controlled experiments with treatment selection and validation. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW, pages 609–618, 2014. doi:[10.1145/2566486.2568028](https://doi.org/10.1145/2566486.2568028).
- [79] Alex Deng, Jiannan Lu, and Shouyuan Chen. Continuous monitoring of A/B tests without pain: Optional stopping in bayesian testing. In *Proceedings of the 3rd International Conference on Data Science and Advanced Analytics*, DSAA, pages 243–252, 2016. doi:[10.1109/dsaa.2016.33](https://doi.org/10.1109/dsaa.2016.33).
- [80] Alex Deng, Jiannan Lu, and Jonthan Litz. Trustworthy analysis of online A/B tests: Pitfalls, challenges and solutions. In *Proceedings of the 10th International Conference on Web Search and Data Mining*, WSDM, pages 641–649, 2017. doi:[10.1145/3018661.3018677](https://doi.org/10.1145/3018661.3018677).

- [81] Pavel Dmitriev and Xian Wu. Measuring metrics. In *Proceedings of the 25th International on Conference on Information and Knowledge Management*, CIKM, pages 429–437, 2016. doi:[10.1145/2983323.2983356](https://doi.org/10.1145/2983323.2983356).
- [82] Pavel Dmitriev, Brian Frasca, Somit Gupta, Ron Kohavi, and Garnet Vaz. Pitfalls of long-term online controlled experiments. In *Proceedings of the 4th International Conference on Big Data*, BigData, pages 1367–1376, 2016. doi:[10.1109/bigdata.2016.7840744](https://doi.org/10.1109/bigdata.2016.7840744).
- [83] Pavel Dmitriev, Somit Gupta, Dong Woo Kim, and Garnet Vaz. A dirty dozen: Twelve common metric interpretation pitfalls in online controlled experiments. In *Proceedings of the 23rd International Conference on Knowledge Discovery and Data Mining*, KDD, pages 1427–1436, 2017. doi:[10.1145/3097983.3098024](https://doi.org/10.1145/3097983.3098024).
- [84] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining*, pages 71–80, 2000. doi:[10.1145/347090.347107](https://doi.org/10.1145/347090.347107).
- [85] Alexey Drutsa. Sign-aware periodicity metrics of user engagement for online search quality evaluation. In *Proceedings of the 38th International Conference on Research and Development in Information Retrieval*, SIGIR, pages 845–854, 2015. doi:[10.1145/2766462.2767814](https://doi.org/10.1145/2766462.2767814).
- [86] Alexey Drutsa, Gleb Gusev, and Pavel Serdyukov. Future user engagement prediction and its application to improve the sensitivity of online experiments. In *Proceedings of the 24th International Conference on World Wide Web*, WWW, pages 256–266, 2015. doi:[10.1145/2736277.2741116](https://doi.org/10.1145/2736277.2741116).
- [87] Alexey Drutsa, Anna Ufliand, and Gleb Gusev. Practical aspects of sensitivity in online experimentation with user engagement metrics. In *Proceedings of the 24th International on Conference on Information and Knowledge Management*, CIKM, pages 763–772, 2015. doi:[10.1145/2806416.2806496](https://doi.org/10.1145/2806416.2806496).
- [88] Alexey Drutsa, Gleb Gusev, and Pavel Serdyukov. Using the delay in a treatment effect to improve sensitivity and preserve directionality of engagement metrics in A/B experiments. In *Proceedings of the 26th International Conference on World Wide Web*, WWW, pages 1301–1310, 2017. doi:[10.1145/3038912.3052664](https://doi.org/10.1145/3038912.3052664).
- [89] Wouter Duivesteijn, Tara Farzami, Thijs Putman, Evertjan Peer, Hilde J. P. Weerts, Jasper N. Adegeest, Gerson Foks, and Mykola Pechenizkiy. Have it both ways: From A/B testing to A&B testing with exceptional model mining. In *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, ECML/PKDD, pages 114–126, 2017. doi:[10.1007/978-3-319-71273-4_10](https://doi.org/10.1007/978-3-319-71273-4_10).

- [90] Tore Dybå and Torgeir Dingsøy. What do we know about agile software development? *IEEE Software*, 26(5):6–9, 2009. doi:[10.1109/MS.2009.145](https://doi.org/10.1109/MS.2009.145).
- [91] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Selecting empirical methods for software engineering research. In Forrest Shull, Janice Singer, and Dag I. K. Sjøberg, editors, *Guide to Advanced Empirical Software Engineering*, pages 285–311. Springer Publishing Company, 2008. doi:[10.1007/978-1-84800-044-5_11](https://doi.org/10.1007/978-1-84800-044-5_11).
- [92] Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. DevOps. *IEEE Software*, 33(3):94–100, 2016. doi:[10.1109/MS.2016.68](https://doi.org/10.1109/MS.2016.68).
- [93] Dean Eckles, Brian Karrer, and Johan Ugander. Design and analysis of experiments in networks: Reducing bias from interference. *Journal of Causal Inference*, 5(1), 2016. doi:[10.1515/jci-2015-0021](https://doi.org/10.1515/jci-2015-0021).
- [94] Ulrik Eklund and Jan Bosch. Architecture for large-scale innovation experiment systems. In *Proceedings of the Joint Working Conference on Software Architecture and European Conference on Software Architecture*, WICSA/ECSCA, pages 244–248, 2012. doi:[10.1109/wicsa-ecsa.212.38](https://doi.org/10.1109/wicsa-ecsa.212.38).
- [95] Sean Ellis. Find a growth hacker for your startup. Accessed: 2022-01-21, 2010. startup-marketing.com/where-are-all-the-growth-hackers.
- [96] Sean Ellis and Morgan Brown. *Hacking growth: how today's fastest-growing companies drive breakout success*. Currency, 2017.
- [97] Adam N. Elmachtoub, Ryan McNellis, Sechan Oh, and Marek Petrik. A practical method for solving contextual bandit problems using decision trees. In *Proceedings of the 33rd Conference on Uncertainty in Artificial Intelligence*, UAI, 2017. arXiv:[1706.04687](https://arxiv.org/abs/1706.04687).
- [98] Emelie Engström, Margaret-Anne Storey, Per Runeson, Martin Höst, and Maria Teresa Baldassarre. How software engineering research aligns with design science: a review. *Empirical Software Engineering*, 25(4):2630–2660, 2020. doi:[10.1007/s10664-020-09818-7](https://doi.org/10.1007/s10664-020-09818-7).
- [99] Aleksander Fabijan, Helena Holmström Olsson Olsson, and Jan Bosch. Customer feedback and data collection techniques in software R&D: A literature review. In *Proceedings of the 6th International Conference on Software Business*, ICSOB, pages 139–153, 2015. doi:[10.1007/978-3-319-19593-3_12](https://doi.org/10.1007/978-3-319-19593-3_12).
- [100] Aleksander Fabijan, Pavel Dmitriev, Helena Holmström Olsson, and Jan Bosch. The benefits of controlled experimentation at scale. In *Proceedings of the 43rd Euromicro Conference on Software Engineering and Advanced Applications*, SEAA, pages 18–26, 2017. doi:[10.1109/SEAA.2017.47](https://doi.org/10.1109/SEAA.2017.47).

- [101] Aleksander Fabijan, Pavel Dmitriev, Helena Holmström Olsson, and Jan Bosch. The evolution of continuous experimentation in software product development: from data to a data-driven organization at scale. In *Proceedings of the 39th International Conference on Software Engineering*, ICSE, pages 770–780, 2017. doi:[10.1109/ICSE.2017.76](https://doi.org/10.1109/ICSE.2017.76).
- [102] Aleksander Fabijan, Pavel Dmitriev, Helena Holmström Olsson, and Jan Bosch. Effective online controlled experiment analysis at large scale. In *Proceedings of the 44th Euromicro Conference on Software Engineering and Advanced Applications*, SEAA, pages 64–67, 2018. doi:[10.1109/seaa.2018.00020](https://doi.org/10.1109/seaa.2018.00020).
- [103] Aleksander Fabijan, Pavel Dmitriev, Helena Holmström Olsson, and Jan Bosch. Online controlled experimentation at scale: An empirical survey on the current state of A/B testing. In *Proceedings of the 44th Euromicro Conference on Software Engineering and Advanced Applications*, SEAA, pages 68–72, 2018. doi:[10.1109/seaa.2018.00021](https://doi.org/10.1109/seaa.2018.00021).
- [104] Aleksander Fabijan, Pavel Dmitriev, Helena Holmström Olsson, and Jan Bosch. The online controlled experiment lifecycle. *IEEE Software*, 37:60–67, 2020. doi:[10.1109/ms.2018.2875842](https://doi.org/10.1109/ms.2018.2875842).
- [105] Fabian Fagerholm, Alejandro Sanchez Guinea, Hanna Mäenpää, and Jürgen Münch. Building blocks for continuous experimentation. In *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*, pages 26–35, 2014. doi:[10.1145/2593812.2593816](https://doi.org/10.1145/2593812.2593816).
- [106] Fabian Fagerholm, Alejandro Sanchez Guinea, Hanna Mäenpää, and Jürgen Münch. The RIGHT model for continuous experimentation. *Journal of Systems and Software*, 123:292–305, 2017. doi:[10.1016/j.jss.2016.03.034](https://doi.org/10.1016/j.jss.2016.03.034).
- [107] Dror G. Feitelson, Eitan Frachtenberg, and Kent L. Beck. Development and deployment at Facebook. *IEEE Internet Computing*, 17(4):8–17, 2013. doi:[10.1109/mic.2013.25](https://doi.org/10.1109/mic.2013.25).
- [108] Alexander Felfernig, Monika Mandl, Juha Tiihonen, Monika Schubert, and Gerhard Leitner. Personalized user interfaces for product configuration. In *Proceedings of the 15th International Conference on Intelligent User Interfaces*, pages 317–320, 2010. doi:[10.1145/1719970.1720020](https://doi.org/10.1145/1719970.1720020).
- [109] Norman E. Fenton and Martin Neil. A critique of software defect prediction models. *IEEE Transactions on software engineering*, 25(5):675–689, 1999. doi:[10.1109/32.815326](https://doi.org/10.1109/32.815326).
- [110] Raphaël Féraud, Robin Allesiardo, Tanguy Urvoy, and Fabrice Clérot. Random forest for the contextual bandit problem. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, PMLR, pages 93–101, 2016.

- [111] Sergio Firmenich, Alejandra Garrido, Julián Grigera, José Matías Rivero, and Gustavo Rossi. Usability improvement through A/B testing and refactoring. *Software Quality Journal*, 27(1):203–240, 2018. doi:[10.1007/s11219-018-9413-y](https://doi.org/10.1007/s11219-018-9413-y).
- [112] Ronald Aylmer Fisher. *The Design of Experiments*. Oliver and Boyd, 10th edition, 1937.
- [113] Brian Fitzgerald and Klaas-Jan Stol. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, 123:176–189, 2017. doi:[10.1016/j.jss.2015.06.063](https://doi.org/10.1016/j.jss.2015.06.063).
- [114] Catherine Flick. Informed consent and the Facebook emotional manipulation study. *Research Ethics*, 12(1):14–28, 2016. doi:[10.1177/1747016115599568](https://doi.org/10.1177/1747016115599568).
- [115] Peter Forbrig. Use cases, user stories and BizDevOps. In *Proceedings of the 4th Workshop on Continuous Requirements Engineering*, CRE, 2018.
- [116] Peter Forbrig and Anke Dittmar. Integrating HCD into BizDevOps by using the subject-oriented approach. In *Proceedings of the International Working Conference on Human-Centred Software Engineering*, HCSE, pages 327–334, 2019. doi:[10.1007/978-3-030-05909-5_21](https://doi.org/10.1007/978-3-030-05909-5_21).
- [117] Alexander Forrester, Andras Sobester, and Andy Keane. *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons, 2008.
- [118] Ilias Gerostathopoulos, Dominik Skoda, Frantisek Plasil, Tomas Bures, and Alessia Knauss. Architectural homeostasis in self-adaptive software-intensive cyber-physical systems. In *Proceedings of the 10th European Conference on Software Architecture*, ECSA, pages 113–128, 2016. doi:[10.1007/978-3-319-48992-6_8](https://doi.org/10.1007/978-3-319-48992-6_8).
- [119] Ilias Gerostathopoulos, Christian Prehofer, Lubomír Bulej, Tomáš Bureš, Vojtech Horký, and Petr Tuma. Cost-aware stage-based experimentation: challenges and emerging results. In *Proceedings of the 15th International Conference on Software Architecture Companion*, ICSA-C, 2018. doi:[10.1109/ICSA-C.2018.00027](https://doi.org/10.1109/ICSA-C.2018.00027).
- [120] Ilias Gerostathopoulos, Christian Prehofer, and Tomas Bures. Adapting a system with noisy outputs with statistical guarantees. In *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS, pages 998–1001, 2018. doi:[10.1145/3194133.3194152](https://doi.org/10.1145/3194133.3194152).
- [121] Ilias Gerostathopoulos, Ali Naci Uysal, Christian Prehofer, and Tomas Bures. A tool for online experiment-driven adaptation. In *Proceedings of the 3rd International Workshops on Foundations and Applications of Self* Systems*, FAS*W, pages 100–105, 2018. doi:[10.1109/FAS-W.2018.00032](https://doi.org/10.1109/FAS-W.2018.00032).

- [122] Federico Giaimo, Hang Yin, Christian Berger, and Ivica Crnkovic. Continuous experimentation on cyber-physical systems: Challenges and opportunities. In *Proceedings of the Scientific Workshops of XP*, pages 1–2, 2016. doi:[10.1145/2962695.2962709](https://doi.org/10.1145/2962695.2962709).
- [123] Federico Giaimo, Christian Berger, and Crispin Kirchner. Considerations about continuous experimentation for resource-constrained platforms in self-driving vehicles. In *Proceedings of the 11th European Conference on Software Architecture*, ECSA, pages 84–91, 2017. doi:[10.1007/978-3-319-65831-5_6](https://doi.org/10.1007/978-3-319-65831-5_6).
- [124] Barney Glaser and Anselm Strauss. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. AldineTransaction, 1967.
- [125] Carlos A. Gomez-Uribe and Neil Hunt. The netflix recommender system. *ACM Transactions on Management Information Systems (TOSEM)*, 6(4):1–19, 2015. doi:[10.1145/2843948](https://doi.org/10.1145/2843948).
- [126] Anjan Goswami, Wei Han, Zhenrui Wang, and Angela Jiang. Controlled experiments for decision-making in e-commerce search. In *Proceedings of the 3rd International Conference on Big Data*, BigData, pages 1094–1102, 2015. doi:[10.1109/bigdata.2015.7363863](https://doi.org/10.1109/bigdata.2015.7363863).
- [127] Colin M. Gray, Yubo Kou, Bryan Battles, Joseph Hoggatt, and Austin L. Toombs. The dark (patterns) side of UX design. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–14, 2018. doi:[10.1145/3173574.3174108](https://doi.org/10.1145/3173574.3174108).
- [128] Huan Gui, Ya Xu, Anmol Bhasin, and Jiawei Han. Network A/B testing. In *Proceedings of the 24th International Conference on World Wide Web*, WWW, page 399–409, 2015. doi:[10.1145/2736277.2741081](https://doi.org/10.1145/2736277.2741081).
- [129] Jayant Gupchup, Yasaman Hosseinkashi, Pavel Dmitriev, Daniel Schneider, Ross Cutler, Andrei Jefremov, and Martin Ellis. Trustworthy experimentation under telemetry loss. In *Proceedings of the 27th International Conference on Information and Knowledge Management*, CIKM, pages 387–396, 2018. doi:[10.1145/3269206.3271747](https://doi.org/10.1145/3269206.3271747).
- [130] Somit Gupta, Lucy Ulanova, Sumit Bhardwaj, Pavel Dmitriev, Paul Raff, and Aleksander Fabijan. The anatomy of a large-scale experimentation platform. In *Proceedings of the 15th International Conference on Software Architecture*, ICSA, pages 1–109, 2018. doi:[10.1109/icsa.2018.00009](https://doi.org/10.1109/icsa.2018.00009).

- [131] Matthias Gutbrod, Jürgen Münch, and Matthias Tichy. How do software startups approach experimentation? empirical results from a qualitative interview study. In *Proceedings of the 18th International Conference on Product-Focused Software Process Improvement*, PROFES, pages 297–304, 2017. doi:[10.1007/978-3-319-69926-4_21](https://doi.org/10.1007/978-3-319-69926-4_21).
- [132] Matthias Gutbrod, Jürgen Münch, and Matthias Tichy. How do software startups approach experimentation? empirical results from a qualitative interview study. In *Proceedings of the 18th International Conference on Product-Focused Software Process Improvement*, PROFES, pages 297–304, 2017. doi:[10.1007/978-3-319-69926-4_21](https://doi.org/10.1007/978-3-319-69926-4_21).
- [133] Irit Hadar, Tomer Hasson, Oshrat Ayalon, Eran Toch, Michael Birnhack, Sofia Sherman, and Arod Balissa. Privacy by designers: software developers’ privacy mindset. *Empirical Software Engineering*, 23(1):259–289, 2018. doi:[10.1007/s10664-017-9517-1](https://doi.org/10.1007/s10664-017-9517-1).
- [134] Mikael Hammar, Robin Karlsson, and Bengt J. Nilsson. Using maximum coverage to optimize recommendation systems in e-commerce. In *Proceedings of the 7th Conference on Recommender Systems*, RecSys, pages 265–272, 2013. doi:[10.1145/2507157.2507169](https://doi.org/10.1145/2507157.2507169).
- [135] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science research in information systems. *MIS Quarterly*, 28(1):75–105, 2004. doi:[10.2307/25148625](https://doi.org/10.2307/25148625).
- [136] Charles Robert Hicks. *Fundamental concepts in the design of experiments*. Holt, Rinehart and Winston, 5th edition, 1964.
- [137] Daniel N. Hill, Robert Moakler, Alan E. Hubbard, Vadim Tsemekhman, Foster Provost, and Kiril Tsemekhman. Measuring causal impact of online actions via natural experiments: Application to display advertising. In *Proceedings of the 21th International Conference on Knowledge Discovery and Data Mining*, KDD, pages 1839–1847, 2015. doi:[10.1145/2783258.2788622](https://doi.org/10.1145/2783258.2788622).
- [138] Daniel N. Hill, Houssam Nassif, Yi Liu, Anand Iyer, and S.V.N. Vishwanathan. An efficient bandit algorithm for realtime multivariate optimization. In *Proceedings of the 23rd International Conference on Knowledge Discovery and Data Mining*, KDD, pages 1813–1821, 2017. doi:[10.1145/3097983.3098184](https://doi.org/10.1145/3097983.3098184).
- [139] Kashmir Hill. Facebook added ‘research’ to user agreement 4 months after emotion manipulation study. *Forbes*, 2014. www.forbes.com/sites/kashmirhill/2014/06/30/facebook-only-got-permission-to-do-research-on-users-after-emotion-manipulation-study.

- [140] Henning Hohnhold, Deirdre O’Brien, and Diane Tang. Focusing on the long-term: It’s good for users and business. In *Proceedings of the 21th International Conference on Knowledge Discovery and Data Mining*, KDD, pages 1849–1858, 2015. doi:[10.1145/2783258.2788583](https://doi.org/10.1145/2783258.2788583).
- [141] Laura M. Holson. Putting a bolder face on Google. *New York Times*, 1, 2009. www.nytimes.com/2009/03/01/business/01marissa.html.
- [142] Holger H. Hoos. Programming by optimization. *Communications of the ACM*, 55(2):70–80, 2012. doi:[10.1145/2076450.2076469](https://doi.org/10.1145/2076450.2076469).
- [143] Chin-Jung Hsu, Vivek Nair, Tim Menzies, and Vincent Freeh. Micky: A cheaper alternative for selecting cloud instances. In *Proceedings of the 11th International Conference on Cloud Computing*, CLOUD, pages 409–416, 2018. doi:[10.1109/CLOUD.2018.00058](https://doi.org/10.1109/CLOUD.2018.00058).
- [144] Xin Huang, He Zhang, Xin Zhou, Muhammad Ali Babar, and Song Yang. Synthesizing qualitative research in software engineering: A critical review. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE, pages 1207–1218, 2018. doi:[10.1145/3180155.3180235](https://doi.org/10.1145/3180155.3180235).
- [145] Jez Humble and Joanne Molesky. Why enterprises must adopt DevOps to enable continuous delivery. *Cutter IT Journal*, 24(8):6, 2011.
- [146] Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, 2014. doi:[10.1016/j.artint.2013.10.003](https://doi.org/10.1016/j.artint.2013.10.003).
- [147] Michael Hüttermann. *DevOps for developers*. Apress, 2012.
- [148] Shuhei Iitsuka and Yutaka Matsuo. Website optimization problem and its solutions. In *Proceedings of the 21th International Conference on Knowledge Discovery and Data Mining*, KDD, pages 447–456, 2015. doi:[10.1145/2783258.2783351](https://doi.org/10.1145/2783258.2783351).
- [149] Deepal Jayasinghe, Josh Kimball, Siddharth Choudhary, Tao Zhu, and Calton Pu. An automated approach to create, store, and analyze large-scale experimental data in clouds. In *Proceedings of the 14th International Conference on Information Reuse and Integration*, IRI, pages 357–364, 2013. doi:[10.1109/iri.2013.6642493](https://doi.org/10.1109/iri.2013.6642493).
- [150] Shan Jiang, John Martin, and Christo Wilson. Who’s the guinea pig? investigating online A/B/n tests in-the-wild. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, FAT*, pages 201–210, 2019. doi:[10.1145/3287560.3287565](https://doi.org/10.1145/3287560.3287565).

- [151] Miguel Jiménez, Luis F. Rivera, Norha M. Villegas, Gabriel Tamura, Hausi A. Müller, and Nelly Bencomo. An architectural framework for quality-driven adaptive continuous experimentation. In *Proceedings of the Joint 4th International Workshop on Rapid Continuous Software Engineering and 1st International Workshop on Data-Driven Decisions, Experimentation and Evolution*, RCoSE/DDrEE, pages 20–23, 2019. doi:[10.1109/RCoSE/DDrEE.2019.00012](https://doi.org/10.1109/RCoSE/DDrEE.2019.00012).
- [152] Jan Ole Johanssen, Anja Kleebaum, Barbara Paech, and Bernd Bruegge. Continuous software engineering and its support by usage and decision knowledge: An interview study with practitioners. *Journal of Software: Evolution and Process*, 31(5):e2169, 2019. doi:[10.1002/smr.2169](https://doi.org/10.1002/smr.2169).
- [153] Ramesh Johari, Pete Koomen, Leonid Pekelis, and David Walsh. Peeking at A/B tests: Why it matters, and what to do about it. In *Proceedings of the 23rd International Conference on Knowledge Discovery and Data Mining*, KDD, pages 1517–1525, 2017. doi:[10.1145/3097983.3097992](https://doi.org/10.1145/3097983.3097992).
- [154] Gerry Johnson, Kevan Scholes, and Richard Whittington. *Exploring corporate strategy: Text and cases*. Pearson Education, 2008.
- [155] Nianqiao Ju, Diane Hu, Adam Henderson, and Liangjie Hong. A sequential test for selecting the better variant: Online A/B testing, adaptive allocation, and continuous monitoring. In *Proceedings of the 12th International Conference on Web Search and Data Mining*, WSDM, pages 492–500, 2019. doi:[10.1145/3289600.3291025](https://doi.org/10.1145/3289600.3291025).
- [156] Gabriela Jurca, Theodore D. Hellmann, and Frank Maurer. Integrating agile and user-centered design: a systematic mapping and review of evaluation and validation studies of agile-ux. In *Proceedings of the Agile Conference*, pages 24–32, 2014. doi:[10.1109/AGILE.2014.17](https://doi.org/10.1109/AGILE.2014.17).
- [157] Narendra Jussien and Olivier Lhomme. Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, 139(1):21–45, 2002. doi:[10.1016/S0004-3702\(02\)00221-7](https://doi.org/10.1016/S0004-3702(02)00221-7).
- [158] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute, 1990.
- [159] Kyo C. Kang, Jaejoon Lee, and Patrick Donohoe. Feature-oriented product line engineering. *IEEE Software*, 19(4):58–65, 2002. doi:[10.1109/MS.2002.1020288](https://doi.org/10.1109/MS.2002.1020288).
- [160] Karim Kanoun and Mihaela van der Schaar. Big-data streaming applications scheduling with online learning and concept drift detection. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition*, DATE, pages 1547–1550, 2015. doi:[10.7873/DATE.2015.0786](https://doi.org/10.7873/DATE.2015.0786).

- [161] Liran Katzir, Edo Liberty, and Oren Somekh. Framework and algorithms for network bucket testing. In *Proceedings of the 21st International Conference on World Wide Web*, WWW, pages 1029–1036, 2012. doi:[10.1145/2187836.2187974](https://doi.org/10.1145/2187836.2187974).
- [162] Kai-Kristian Kemell, Polina Feshchenko, Joonas Himmanen, Abrar Hossain, Furqan Jameel, Raffaele Luigi Puca, Teemu Vitikainen, Joni Kultanen, Juhani Risku, Johannes Impiö, et al. Software startup education: gamifying growth hacking. In *Proceedings of the 2nd International Workshop on Software-Intensive Business: Start-ups, Platforms, and Ecosystems*, IWSiB, pages 25–30, 2019. doi:[10.1145/3340481.3342734](https://doi.org/10.1145/3340481.3342734).
- [163] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003. doi:[10.1109/MC.2003.1160055](https://doi.org/10.1109/MC.2003.1160055).
- [164] Katja Kevic, Brendan Murphy, Laurie Williams, and Jennifer Beckmann. Characterizing experimentation in continuous deployment: a case study on Bing. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track*, ICSE-SEIP, pages 123–132, 2017. doi:[10.1109/ICSE-SEIP.2017.19](https://doi.org/10.1109/ICSE-SEIP.2017.19).
- [165] Eugene Kharitonov, Craig Macdonald, Pavel Serdyukov, and Iadh Ounis. Optimised scheduling of online experiments. In *Proceedings of the 38th International Conference on Research and Development in Information Retrieval*, SIGIR, pages 453–462, 2015. doi:[10.1145/2766462.2767706](https://doi.org/10.1145/2766462.2767706).
- [166] Eugene Kharitonov, Aleksandr Vorobev, Craig Macdonald, Pavel Serdyukov, and Iadh Ounis. Sequential testing for early stopping of online experiments. In *Proceedings of the 38th International Conference on Research and Development in Information Retrieval*, SIGIR, pages 473–482, 2015. doi:[10.1145/2766462.2767729](https://doi.org/10.1145/2766462.2767729).
- [167] Eugene Kharitonov, Alexey Drutsa, and Pavel Serdyukov. Learning sensitive combinations of A/B test metrics. In *Proceedings of the 10th International Conference on Web Search and Data Mining*, WSDM, pages 651–659, 2017. doi:[10.1145/3018661.3018708](https://doi.org/10.1145/3018661.3018708).
- [168] Jacqueline Kirtley and Siobhan O’Mahony. What is a pivot? explaining when and how entrepreneurial firms decide to make strategic change and pivot. *Strategic Management Journal*, Special Issue, 2020. doi:[10.1002/smj.3131](https://doi.org/10.1002/smj.3131).
- [169] Barbara Ann Kitchenham, Tore Dyba, and Magne Jorgensen. Evidence-based software engineering. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE, pages 273–281, 2004. doi:[10.1109/ICSE.2004.1317449](https://doi.org/10.1109/ICSE.2004.1317449).

- [170] Ron Kohavi and Roger Longbotham. Unexpected results in online controlled experiments. *ACM SIGKDD Explorations Newsletter*, 12(2):31, 2011. doi:[10.1145/1964897.1964905](https://doi.org/10.1145/1964897.1964905).
- [171] Ron Kohavi, Randal M. Henne, and Dan Sommerfield. Practical guide to controlled experiments on the web: Listen to your customers not to the HiPPO. In *Proceedings of the 13th International Conference on Knowledge Discovery and Data Mining*, KDD, pages 959–967, 2007. doi:[10.1145/1281192.1281295](https://doi.org/10.1145/1281192.1281295).
- [172] Ron Kohavi, Roger Longbotham, Dan Sommerfield, and Randal M. Henne. Controlled experiments on the web: Survey and practical guide. *Data Mining and Knowledge Discovery*, 18(1):140–181, 2009. doi:[10.1007/s10618-008-0114-1](https://doi.org/10.1007/s10618-008-0114-1).
- [173] Ron Kohavi, Alex Deng, Brian Frasca, Roger Longbotham, Toby Walker, and Ya Xu. Trustworthy online controlled experiments: Five puzzling outcomes explained. In *Proceedings of the 18th International Conference on Knowledge Discovery and Data Mining*, KDD, pages 786–794, 2012. doi:[10.1145/2339530.2339653](https://doi.org/10.1145/2339530.2339653).
- [174] Ron Kohavi, Alex Deng, Brian Frasca, Toby Walker, Ya Xu, and Nils Pohlmann. Online controlled experiments at large scale. In *Proceedings of the 19th International Conference on Knowledge Discovery and Data Mining*, KDD, pages 1168–1176, 2013. doi:[10.1145/2487575.2488217](https://doi.org/10.1145/2487575.2488217).
- [175] Ron Kohavi, Alex Deng, Roger Longbotham, and Ya Xu. Seven rules of thumb for web site experimenters. In *Proceedings of the 20th International Conference on Knowledge Discovery and Data Mining*, KDD, pages 1857–1866, 2014. doi:[10.1145/2623330.2623341](https://doi.org/10.1145/2623330.2623341).
- [176] Ronny Kohavi, Thomas Crook, Roger Longbotham, Brian Frasca, Randy Henne, Juan Lavista Ferres, and Tamir Melamed. Online experimentation at Microsoft. *Proceedings of the 3rd International Workshop on Data Mining Case Studies*, 11, 2009.
- [177] Kostantinos Koukouvis, Roberto Alcañiz Cubero, and Patrizio Pelliccione. A/B testing in e-commerce sales processes. In *Proceedings of the 8th International Workshop on Software Engineering for Resilient Systems*, SERENE, pages 133–148, 2016. doi:[10.1007/978-3-319-45892-2_10](https://doi.org/10.1007/978-3-319-45892-2_10).
- [178] John F. Krafcik. Triumph of the lean production system. *Sloan Management Review*, 30(1):41–52, 1988.
- [179] Adam D. I. Kramer, Jamie E. Guillory, and Jeffrey T. Hancock. Experimental evidence of massive-scale emotional contagion through social networks. *Proceedings of the National Academy of Sciences of the United States of America (PNAS)*, 111(24): 8788–8790, 2014. doi:[10.1073/pnas.1320040111](https://doi.org/10.1073/pnas.1320040111).

- [180] Krzysztof Kuchcinski. Constraints-driven scheduling and resource assignment. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 8(3):355–383, 2003. doi:[10.1145/785411.785416](https://doi.org/10.1145/785411.785416).
- [181] Bill Kuechler and Vijay Vaishnavi. On theory development in design science research: anatomy of a research project. *European Journal of Information Systems*, 17(5):489–504, 2008. doi:[10.1057/ejis.2008.40](https://doi.org/10.1057/ejis.2008.40).
- [182] Terran Lane and Carla E. Brodley. Approaches to online learning and concept drift for user identification in computer security. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining, KDD*, pages 259–263, 1998.
- [183] Minyong R. Lee and Milan Shen. Winner’s curse: Bias estimation for total effects of features in online controlled experiments. In *Proceedings of the 24th International Conference on Knowledge Discovery and Data Mining, KDD*, pages 491–499, 2018. doi:[10.1145/3219819.3219905](https://doi.org/10.1145/3219819.3219905).
- [184] Benjamin Letham, Brian Karrer, Guilherme Ottoni, and Eytan Bakshy. Constrained bayesian optimization with noisy experiments. *Bayesian Analysis*, 14(2):495–519, 2019. doi:[10.1214/18-ba1110](https://doi.org/10.1214/18-ba1110).
- [185] Florian Lettner, Clemens Holzmann, and Patrick Hutflasz. Enabling A/B testing of native mobile applications by remote user interface exchange. In *Proceedings of the 14th International Conference on Computer Aided Systems Theory, EUROCAST*, pages 458–466, 2013. doi:[10.1007/978-3-642-53862-9_58](https://doi.org/10.1007/978-3-642-53862-9_58).
- [186] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web, WWW*, pages 661–670, 2010. doi:[10.1145/1772690.1772758](https://doi.org/10.1145/1772690.1772758).
- [187] Greg Linden. Marissa Mayer at web 2.0. Accessed: 2022-01-24, 2006. glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html.
- [188] Eveliina Lindgren and Jürgen Münch. Raising the odds of success: The current state of experimentation in product development. *Information and Software Technology*, 77:80–91, 2016. doi:[10.1016/j.infsof.2016.04.008](https://doi.org/10.1016/j.infsof.2016.04.008).
- [189] C. H. Liu, Ângelo Cardoso, Paul Couturier, and Emma J. McCoy. Datasets for online controlled experiments. In *Proceedings of the 35th International Conference on Neural Information Processing Systems: Datasets and Benchmarks Track, NeurIPS*, 2021. arXiv:[2111.10198](https://arxiv.org/abs/2111.10198).

- [190] Luo Lu and Chuang Liu. Separation strategies for three pitfalls in A/B testing. In *Proceedings of the 2nd KDD Workshop on User Engagement Optimization*, UEO, pages 1–7, 2014.
- [191] Roman Lukyanenko, Joerg Evermann, and Jeffrey Parsons. Instantiation validity in IS design research. In *Proceedings of the 9th International Conference on Design Science Research in Information Systems*, DESRIST, pages 321–328, 2014.
- [192] Alan MacCormack, John Rusnak, and Carliss Y. Baldwin. Exploring the structure of complex software designs: An empirical study of open source and proprietary code. *Management Science*, 52(7):1015–1030, 2006. doi:[10.1287/mnsc.1060.0552](https://doi.org/10.1287/mnsc.1060.0552).
- [193] Widad Machmouchi and Georg Buscher. Principles for the design of online A/B metrics. In *Proceedings of the 39th International Conference on Research and Development in Information Retrieval*, SIGIR, pages 589–590, 2016. doi:[10.1145/2911451.2926731](https://doi.org/10.1145/2911451.2926731).
- [194] Widad Machmouchi, Ahmed Hassan Awadallah, Imed Zitouni, and Georg Buscher. Beyond success rate: Utility as a search quality metric for online experiments. In *Proceedings of the 26th on Conference on Information and Knowledge Management*, CIKM, pages 757–765, 2017. doi:[10.1145/3132847.3132850](https://doi.org/10.1145/3132847.3132850).
- [195] Satoshi Masuda, Kohichi Ono, Toshiaki Yasue, and Nobuhiro Hosokawa. A survey of software quality for machine learning applications. In *Proceedings of the 11th International Conference on Software Testing, Verification and Validation Workshops*, ICSTW, pages 279–284, 2018. doi:[10.1109/ICSTW.2018.00061](https://doi.org/10.1109/ICSTW.2018.00061).
- [196] David Issa Mattos, Jan Bosch, and Helena Holmström Olsson. Your system gets better every day you use it: Towards automated continuous experimentation. In *Proceedings of the 43rd Euromicro Conference on Software Engineering and Advanced Applications*, SEAA, pages 256–265, 2017. doi:[10.1109/SEAA.2017.15](https://doi.org/10.1109/SEAA.2017.15).
- [197] David Issa Mattos, Jan Bosch, and Helena Holmström Olsson. Challenges and strategies for undertaking continuous experimentation to embedded systems: Industry and research perspectives. In *Proceedings of the 19th International Conference on Agile Processes in Software Engineering and Extreme Programming*, XP, pages 277–292, 2018. doi:[10.1007/978-3-319-91602-6_20](https://doi.org/10.1007/978-3-319-91602-6_20).
- [198] David Issa Mattos, Pavel Dmitriev, Aleksander Fabijan, Jan Bosch, and Helena Holmström Olsson. An activity and metric model for online controlled experiments. In *Proceedings of the 19th International Conference on Product-Focused Software Process Improvement*, PROFES, pages 182–198, 2018. doi:[10.1007/978-3-030-03673-7_14](https://doi.org/10.1007/978-3-030-03673-7_14).

- [199] David Issa Mattos, Erling Mårtensson, Jan Bosch, and Helena Holmström Olsson. Optimization experiments in the continuous space. In *Proceedings of the 10th International Symposium on Search-Based Software Engineering*, SSBSE, pages 293–308, 2018. doi:[10.1007/978-3-319-99241-9_16](https://doi.org/10.1007/978-3-319-99241-9_16).
- [200] David Issa Mattos, Jan Bosch, and Helena Holmström Olsson. Multi-armed bandits in the wild: Pitfalls and strategies in online experiments. *Information and Software Technology*, 113:68–81, 2019. doi:[10.1016/j.infsof.2019.05.004](https://doi.org/10.1016/j.infsof.2019.05.004).
- [201] Ash Maurya. *Running lean: iterate from plan A to a plan that works*. O’Reilly Media, 2012.
- [202] Dave McClure. Startup metrics for pirates: AARRR! Accessed: 2022-01-15, 2007. 500hats.typepad.com/500blogs/2007/09/startup-metrics.html.
- [203] Dan McKinley. Design for continuous experimentation: Talk and slides. Accessed: 2019-08-01, 2012. mcfunley.com/design-for-continuous-experimentation.
- [204] Andrés Muñoz Medina, Sergei Vassilvitskii, and Dong Yin. Online learning for non-stationary A/B tests. In *Proceedings of the 27th International Conference on Information and Knowledge Management*, CIKM, pages 317–326, 2018. doi:[10.1145/3269206.3271718](https://doi.org/10.1145/3269206.3271718).
- [205] Jorge Melegati, Xiaofeng Wang, and Pekka Abrahamsson. Hypotheses engineering: first essential steps of experiment-driven software development. In *Proceedings of the Joint 4th International Workshop on Rapid Continuous Software Engineering and 1st International Workshop on Data-Driven Decisions, Experimentation and Evolution*, RCoSE/DDrEE, pages 16–19, 2019. doi:[10.1109/RCoSE/DDrEE.2019.00011](https://doi.org/10.1109/RCoSE/DDrEE.2019.00011).
- [206] Jorge Melegati, Henry Edison, and Xiaofeng Wang. XPro: a model to explain the limited adoption and implementation of experimentation in software startups. *IEEE Transactions on Software Engineering*, Early Access, 2020. doi:[10.1109/TSE.2020.3042610](https://doi.org/10.1109/TSE.2020.3042610).
- [207] Robinson Meyer. Everything we know about Facebook’s secret mood-manipulation experiment. *The Atlantic*, 28, 2014. www.theatlantic.com/technology/archive/2014/06/everything-we-know-about-facebooks-secret-mood-manipulation-experiment/373648.
- [208] Risto Miikkulainen, Gurmeet Lamba, Neil Iscoe, Aaron Shagrin, Ron Cordell, Sam Nazari, Cory Schoolland, Myles Brundage, Jonathan Epstein, and Randy Dean. Conversion rate optimization through evolutionary computation. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO, pages 1193–1199, 2017. doi:[10.1145/3071178.3071312](https://doi.org/10.1145/3071178.3071312).

- [209] Risto Miikkulainen, Neil Iscoe, Aaron Shagrin, Ryan Rapp, Sam Nazari, Patrick McGrath, Cory Schoolland, Elyas Achkar, Myles Brundage, Jeremy Miller, et al. Sentient Ascend: AI-based massively multivariate conversion rate optimization. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, 2018.
- [210] Leandro L. Minku and Xin Yao. DDD: A new ensemble approach for dealing with concept drift. *IEEE Transactions on Knowledge and Data Engineering*, 24(4):619–633, 2011. doi:[10.1109/TKDE.2011.58](https://doi.org/10.1109/TKDE.2011.58).
- [211] Rahul Mohanani, Iftaah Salman, Burak Turhan, Pilar Rodríguez, and Paul Ralph. Cognitive biases in software engineering: a systematic mapping study. *IEEE Transactions on Software Engineering*, 46(12):1318–1339, 2018. doi:[10.1109/TSE.2018.2877759](https://doi.org/10.1109/TSE.2018.2877759).
- [212] Caique Moreira and Breno de França. Towards a healthier collaboration at the business-development interface. In *Proceedings of the 10th Ibero-American Conference on Software Engineering*, CIBSE, pages 86–99, 2019.
- [213] Hussan Munir, Per Runeson, and Krzysztof Wnuk. A theory of openness for software engineering tools in software organizations. *Information and Software Technology*, 97: 26–45, 2018. doi:[10.1016/j.infsof.2017.12.008](https://doi.org/10.1016/j.infsof.2017.12.008).
- [214] Vivek Nair, Zhe Yu, Tim Menzies, Norbert Siegmund, and Sven Apel. Finding faster configurations using FLASH. *IEEE Transactions on Software Engineering*, 46, 2018. doi:[10.1109/TSE.2018.2870895](https://doi.org/10.1109/TSE.2018.2870895).
- [215] Luigi Nardi, David Koeplinger, and Kunle Olukotun. Practical design space exploration. In *Proceedings of the 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, MASCOTS, pages 347–358, 2019. doi:[10.1109/MASCOTS.2019.00045](https://doi.org/10.1109/MASCOTS.2019.00045).
- [216] Irina Niculescu, Huibin Mary Hu, Christina Gee, Chewy Chong, Shivam Dubey, and Paul Luo Li. Towards inclusive software engineering through A/B testing: A case-study at Windows. In *Proceedings of the 43rd International Conference on Software Engineering: Software Engineering in Practice Track*, ICSE-SEIP, pages 180–187, 2021. doi:[10.1109/ICSE-SEIP52600.2021.00027](https://doi.org/10.1109/ICSE-SEIP52600.2021.00027).
- [217] Marius Florin Niculescu and Dong Jun Wu. Economics of free under perpetual licensing: Implications for the software industry. *Information Systems Research*, 25(1):173–199, 2014. doi:[10.2139/ssrn.1853603](https://doi.org/10.2139/ssrn.1853603).
- [218] Michael Nolting and Jan Eike von Seggern. Context-based A/B test validation. In *Proceedings of the 25th International Conference Companion on World Wide Web*, WWW, pages 277–278, 2016. doi:[10.1145/2872518.2889306](https://doi.org/10.1145/2872518.2889306).

- [219] Dan Olsen. *The lean product playbook: How to innovate with minimum viable products and rapid customer feedback*. John Wiley & Sons, 2015.
- [220] Helena Holmström Olsson and Jan Bosch. From opinions to data-driven software R&D: A multi-case study on how to close the ‘open loop’ problem. In *Proceedings of the 40th Euromicro Conference on Software Engineering and Advanced Applications*, SEAA, pages 9–16, 2014. doi:[10.1109/seaa.2014.75](https://doi.org/10.1109/seaa.2014.75).
- [221] Helena Holmström Olsson and Jan Bosch. The HYPEX model: from opinions to data-driven software development. In Jan Bosch, editor, *Continuous Software Engineering*, pages 155–164. Springer Publishing Company, 2014. doi:[10.1007/978-3-319-11283-1_13](https://doi.org/10.1007/978-3-319-11283-1_13).
- [222] Helena Holmström Olsson and Jan Bosch. Data driven development: Challenges in online, embedded and on-premise software. In *Proceedings of the 20th International Conference on Product-Focused Software Process Improvement*, PROFES, pages 515–527, 2019.
- [223] Helena Holmström Olsson, Hiva Alahyari, and Jan Bosch. Climbing the “stairway to heaven”: A multiple-case study exploring barriers in the transition from agile development towards continuous deployment of software. In *Proceedings of the 38th Euromicro Conference on Software Engineering and Advanced Applications*, SEAA, pages 392–399, 2012. doi:[10.1109/SEAA.2012.54](https://doi.org/10.1109/SEAA.2012.54).
- [224] Helena Holmström Olsson, Jan Bosch, and Aleksander Fabijan. Experimentation that matters: A multi-case study on the challenges with A/B testing. In *Proceedings of the 8th International Conference on Software Business*, ICSOB, pages 179–185, 2017. doi:[10.1007/978-3-319-69191-6_12](https://doi.org/10.1007/978-3-319-69191-6_12).
- [225] Alexander Osterwalder. *The business model ontology a proposition in a design science approach*. PhD thesis, Faculty of Business and Economics of the University of Lausanne, 2004.
- [226] Alexander Osterwalder and Yves Pigneur. *Business model generation: A handbook for visionaries, game changers, and challengers*, volume 1. John Wiley & Sons, 2010.
- [227] Jakob Pennington. The eight phases of a DevOps pipeline: Introduction to DevOps part 2. Accessed: 2022-01-15, 2019. <https://medium.com/taptuit/the-eight-phases-of-a-devops-pipeline-fda53ec9bba>.
- [228] Pulasthi Perera, Roshali Silva, and Indika Perera. Improve software quality through practicing DevOps. In *Proceedings of the 17th International Conference on Advances in ICT for Emerging Regions*, ICTer, pages 1–6, 2017. doi:[10.1109/ICTER.2017.8257807](https://doi.org/10.1109/ICTER.2017.8257807).

- [229] Tekla S. Perry. Marissa mayer: Google’s chic geek: This self-proclaimed “girly girl” runs one of google’s fastest-growing services. *IEEE Spectrum*, 49(4):32–64, 2012. doi:[10.1109/MSPEC.2012.6172806](https://doi.org/10.1109/MSPEC.2012.6172806).
- [230] Kai Petersen and Claes Wohlin. Context in industrial software engineering research. In *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement*, ESEM, pages 401–404, 2009. doi:[10.1109/ESEM.2009.5316010](https://doi.org/10.1109/ESEM.2009.5316010).
- [231] Alexander Peysakhovich and Dean Eckles. Learning causal effects from many randomized experiments using regularized instrumental variables. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, WWW, pages 699–707, 2018. doi:[10.1145/3178876.3186151](https://doi.org/10.1145/3178876.3186151).
- [232] Klaus Pohl. *Requirements Engineering: Fundamentals, Principles, and Techniques*. Springer Publishing Company, 2010.
- [233] Klaus Pohl, Günter Böckle, and Frank van Der Linden. *Software product line engineering: foundations, principles, and techniques*. Springer Publishing Company, 2005.
- [234] Alexey Poyarkov, Alexey Drutsa, Andrey Khalyavin, Gleb Gusev, and Pavel Serdyukov. Boosted decision tree regression adjustment for variance reduction in online controlled experiments. In *Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining*, KDD, pages 235–244, 2016. doi:[10.1145/2939672.2939688](https://doi.org/10.1145/2939672.2939688).
- [235] Juan Qin, Wei Qi, and Baojian Zhou. Research on optimal selection strategy of search engine keywords based on multi-armed bandit. In *Proceedings of the 49th Hawaii International Conference on System Sciences*, HICSS, pages 726–734, 2016.
- [236] Md Tajmilur Rahman, Louis-Philippe Querel, Peter C. Rigby, and Bram Adams. Feature toggles: practitioner practices and a case study. In *Proceedings of the 13th International Workshop on Mining Software Repositories*, MSR, pages 201–211, 2016. doi:[10.1145/2901739.2901745](https://doi.org/10.1145/2901739.2901745).
- [237] Reza Rahutomo, Yulius Lie, Anzaludin Samsinga Perbangsa, and Bens Pardamean. Improving conversion rates for fashion e-commerce with A/B testing. In *Proceedings of the 2020 International Conference on Information Management and Technology*, ICIMTech, pages 266–270, 2020. doi:[10.1109/ICIMTech50083.2020.9210947](https://doi.org/10.1109/ICIMTech50083.2020.9210947).
- [238] Risto Rajala, Matti Rossi, and Virpi Kristiina Tuunainen. A framework for analyzing software business models. In *Proceedings of the 11th European Conference on Information Systems*, ECIS, pages 1614–1627, 2003.

- [239] Á Révész and Norbert Pataki. Containerized A/B testing. In *Proceedings of the 6th Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications*, SQAMIA, page 14, 2017.
- [240] Eric Ries. *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. Crown Business, 2011.
- [241] Carlos Riquelme, George Tucker, and Jasper Snoek. Deep bayesian bandits showdown: An empirical comparison of Bayesian deep networks for Thompson sampling. In *Proceedings of the 6th International Conference on Learning Representations*, ICLR, 2018. arXiv:[1802.09127](https://arxiv.org/abs/1802.09127).
- [242] Olli Rissanen and Jürgen Münch. Continuous experimentation in the B2B domain: a case study. In *Proceedings of the 2nd International Workshop on Rapid Continuous Software Engineering*, RCoSE, pages 12–18, 2015. doi:[10.1109/RCoSE.2015.10](https://doi.org/10.1109/RCoSE.2015.10).
- [243] Pilar Rodriguez, Cathy Urquhart, and Emilia Mendes. A theory of value for value-based feature selection in software engineering. *IEEE Transactions on Software Engineering*, Early Access, 2020. doi:[10.1109/TSE.2020.2989666](https://doi.org/10.1109/TSE.2020.2989666).
- [244] Elisabetta Ronchieri and M. Canaparo. Metrics for software reliability: a systematic mapping study. *Journal of Integrated Design and Process Science*, 22(2):5–25, 2018. doi:[10.3233/jid-2018-0008](https://doi.org/10.3233/jid-2018-0008).
- [245] Rasmus Ros. Continuous experimentation interview instrument, 2017. serg.cs.lth.se/fileadmin/serg/Continuous_Experimentation_Instrument.pdf.
- [246] Rasmus Ros. Continuous experimentation with product-led business models: A comparative case study. In *Proceedings of the 11th International Conference on Software Business*, ICSOB, pages 143–158, 2020. doi:[10.1007/978-3-030-67292-8_11](https://doi.org/10.1007/978-3-030-67292-8_11).
- [247] Rasmus Ros and Elizabeth Bjarnason. Continuous experimentation scenarios: A case study in e-commerce. In *Proceedings of the 44th Euromicro Conference on Software Engineering and Advanced Applications*, SEAA, pages 353–356, 2018. doi:[10.1109/seaa.2018.00064](https://doi.org/10.1109/seaa.2018.00064).
- [248] Rasmus Ros and Mikael Hammar. Data-driven software design with constraint oriented multi-variate bandit optimization (COMBO). *Empirical Software Engineering*, 25(5):3841–3872, 2020. doi:[10.1007/s10664-020-09856-1](https://doi.org/10.1007/s10664-020-09856-1).
- [249] Rasmus Ros and Per Runeson. Continuous experimentation and A/B testing: A mapping study. In *Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering*, RCoSE, pages 35–41, 2018. doi:[10.1145/3194760.3194766](https://doi.org/10.1145/3194760.3194766).

- [250] Rasmus Ros, Elizabeth Bjarnason, and Per Runeson. Automated controlled experimentation on software by evolutionary bandit optimization. In *Proceedings of the 9th International Symposium on Search Based Software Engineering*, SSBSE, pages 190–196, 2017. doi:[10.1007/978-3-319-66299-2_18](https://doi.org/10.1007/978-3-319-66299-2_18).
- [251] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.
- [252] Ranjit K. Roy. *Design of Experiments Using the Taguchi Approach: 16 Steps to Product and Process Improvement*. John Wiley & Sons, 2001.
- [253] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009. doi:[10.1007/s10664-008-9102-8](https://doi.org/10.1007/s10664-008-9102-8).
- [254] Per Runeson, Martin Höst, Austen Rainer, and Björn Regnell. *Case study research in software engineering: Guidelines and examples*. John Wiley & Sons, 2012.
- [255] Per Runeson, Emelie Engström, and Margaret-Anne Storey. The design science paradigm as a frame for empirical software engineering. In Michael Felderer and Guilherme Horta Travassos, editors, *Contemporary Empirical Methods in Software Engineering*, pages 127–147. Springer Publishing Company, 2020. doi:[10.1007/978-3-030-32489-6_5](https://doi.org/10.1007/978-3-030-32489-6_5).
- [256] Daniel J. Russo, Benjamin van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen. A tutorial on thompson sampling. *Foundations and Trends in Machine Learning*, 11(1):1–96, 2017. doi:[10.1561/22000000070](https://doi.org/10.1561/22000000070).
- [257] Dina Salah, Richard F. Paige, and Paul Cairns. A systematic literature review for agile development processes and user centred design integration. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE, pages 1–10, 2014. doi:[10.1145/2601248.2601276](https://doi.org/10.1145/2601248.2601276).
- [258] Jeff Sauro and James R. Lewis. *Quantifying the user experience: Practical statistics for user research*. Morgan Kaufmann Publishers, 2016. doi:[10.1016/C2010-0-65192-3](https://doi.org/10.1016/C2010-0-65192-3).
- [259] Tanja Sauvola, Markus Kelanti, Jarkko Hyysalo, Pasi Kuvaja, and Kari Liukkonen. Continuous improvement and validation with customer touchpoint model in software development. In *Proceedings of the 13th International Conference on Software Engineering Advances*, ICSEA, pages 52–60, 2018.
- [260] Martin Saveski, Jean Pouget-Abadie, Guillaume Saint-Jacques, Weitao Duan, Souvik Ghosh, Ya Xu, and Edoardo M. Airoldi. Detecting network effects. In *Proceedings of the 23rd International Conference on Knowledge Discovery and Data Mining*, KDD, pages 1027–1035, 2017. doi:[10.1145/3097983.3098192](https://doi.org/10.1145/3097983.3098192).

- [261] Andrew I. Schein, Alexandrin Popescul, Lyle H. Ungar, and David M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th International Conference on Research and Development in Information Retrieval*, pages 253–260, 2002. doi:[10.1145/564376.564421](https://doi.org/10.1145/564376.564421).
- [262] Gerald Schermann and Philipp Leitner. Search-based scheduling of experiments in continuous deployment. In *Proceedings of the 34th International Conference on Software Maintenance and Evolution*, ICSME, pages 485–495, 2018. doi:[10.1109/icsme.2018.00059](https://doi.org/10.1109/icsme.2018.00059).
- [263] Gerald Schermann, Dominik Schöni, Philipp Leitner, and Harald C. Gall. Bifrost – supporting continuous deployment with automated enactment of multi-phase live testing strategies. In *Proceedings of the 17th International Conference on Middleware*, Middleware, pages 1–14, 2016. doi:[10.1145/2988336.2988348](https://doi.org/10.1145/2988336.2988348).
- [264] Gerald Schermann, Jürgen Cito, and Philipp Leitner. Continuous experimentation: challenges, implementation techniques, and current research. *IEEE Software*, 35(2): 26–31, 2018. doi:[10.1109/MS.2018.111094748](https://doi.org/10.1109/MS.2018.111094748).
- [265] Gerald Schermann, Jürgen Cito, Philipp Leitner, Uwe Zdun, and Harald C. Gall. We’re doing it live: A multi-method empirical study on continuous experimentation. *Information and Software Technology*, 99:41–57, 2018. doi:[10.1016/j.infsof.2018.02.010](https://doi.org/10.1016/j.infsof.2018.02.010).
- [266] Markus Schief and Peter Buxmann. Business models in the software industry. In *Proceedings of the 45th Hawaii International Conference on System Sciences*, HICSS, pages 3328–3337, 2012. doi:[10.1109/HICSS.2012.140](https://doi.org/10.1109/HICSS.2012.140).
- [267] Robert Schumacher. *The handbook of global user research*. Morgan Kaufmann Publishers, 2009.
- [268] Daniel Schwabe, R. Mattos Guimarães, and Gustavo Rossi. Cohesive design of personalized web applications. *IEEE Internet Computing*, 6(2):34–43, 2002. doi:[10.1109/4236.991441](https://doi.org/10.1109/4236.991441).
- [269] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, and Michael Young. Machine learning: The high interest credit card of technical debt. In *Proceedings of the 1st NIPS Workshop on Software Engineering for Machine Learning*, SE4ML, 2014.
- [270] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access*, 5:3909–3943, 2017. doi:[10.1109/ACCESS.2017.2685629](https://doi.org/10.1109/ACCESS.2017.2685629).

- [271] José L. Silva, José Creissac Campos, and Ana C. R. Paiva. Model-based user interface testing with spec explorer and ConcurTaskTrees. *Electronic Notes in Theoretical Computer Science*, 208:77–93, 2008. doi:[10.1016/j.entcs.2008.03.108](https://doi.org/10.1016/j.entcs.2008.03.108).
- [272] Herbert A. Simon. *The Sciences of the Artificial*. MIT Press, 1969.
- [273] Janice Singer, Susan E. Sim, and Timothy C. Lethbridge. Software engineering data collection for field studies. In Forrest Shull, Janice Singer, and Dag I. K. Sjøberg, editors, *Guide to Advanced Empirical Software Engineering*, pages 9–34. Springer Publishing Company, 2008. doi:[10.1007/978-1-84800-044-5_1](https://doi.org/10.1007/978-1-84800-044-5_1).
- [274] Dag I. K. Sjøberg, Tore Dybå, Bente C. D. Anda, and Jo E. Hannay. Building theories in software engineering. In Forrest Shull, Janice Singer, and Dag I. K. Sjøberg, editors, *Guide to Advanced Empirical Software Engineering*, pages 312–336. Springer Publishing Company, 2008. doi:[10.1007/978-1-84800-044-5_12](https://doi.org/10.1007/978-1-84800-044-5_12).
- [275] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Proceedings of the 26th International Conference on Advances in Neural Information Processing Systems*, NIPS, pages 2951–2959, 2012.
- [276] Alina Sorescu. Data-driven business model innovation. *Journal of Product Innovation Management*, 34(5):691–696, 2017. doi:[10.1111/jpim.12398](https://doi.org/10.1111/jpim.12398).
- [277] Maximilian Speicher, Andreas Both, and Martin Gaedke. Ensuring web interface quality through usability-based split testing. In *Proceedings of the 14th International Conference on Web Engineering*, ICWE, pages 93–110, 2014. doi:[10.1007/978-3-319-08245-5_6](https://doi.org/10.1007/978-3-319-08245-5_6).
- [278] Klaas-Jan Stol and Brian Fitzgerald. Theory-oriented software engineering. *Science of Computer Programming*, 101:79–98, 2015. doi:[10.1016/j.scico.2014.11.010](https://doi.org/10.1016/j.scico.2014.11.010).
- [279] Klaas-Jan Stol and Brian Fitzgerald. The ABC of software engineering research. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 27(3):1–51, 2018. doi:doi.org/10.1145/3241743.
- [280] Klaas-Jan Stol and Brian Fitzgerald. Guidelines for conducting software engineering research. In Michael Felderer and Guilherme Horta Travassos, editors, *Contemporary Empirical Methods in Software Engineering*, pages 27–62. Springer Publishing Company, 2020. doi:[10.1007/978-3-030-32489-6_2](https://doi.org/10.1007/978-3-030-32489-6_2).
- [281] Klaas-Jan Stol, Paul Ralph, and Brian Fitzgerald. Grounded theory in software engineering research: a critical review and guidelines. In *Proceedings of the 38th International Conference on Software Engineering*, pages 120–131, 2016. doi:[10.1145/2884781.2884833](https://doi.org/10.1145/2884781.2884833).

- [282] Margaret-Anne Storey, Emelie Engstrom, Martin Höst, Per Runeson, and Elizabeth Bjarnason. Using a visual abstract as a lens for communicating and promoting design science research in software engineering. In *Proceedings of the 11th International Symposium on Empirical Software Engineering and Measurement*, ESEM, pages 181–186, 2017. doi:[10.1109/ESEM.2017.28](https://doi.org/10.1109/ESEM.2017.28).
- [283] Jianyong Sun, Hu Zhang, Aimin Zhou, Qingfu Zhang, Ke Zhang, Zhenbiao Tu, and Kai Ye. Learning from a stream of nonstationary and dependent data in multiobjective evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 23(4):541–555, 2018. doi:[10.1109/TEVC.2018.2865495](https://doi.org/10.1109/TEVC.2018.2865495).
- [284] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 2nd edition, 1998.
- [285] Giordano Tamburrelli and Alessandro Margara. Towards automated A/B testing. In *Proceedings of the 6th International Symposium on Search-Based Software Engineering*, SSBSE, pages 184–198, 2014. doi:[10.1007/978-3-319-09940-8_13](https://doi.org/10.1007/978-3-319-09940-8_13).
- [286] Diane Tang, Ashish Agarwal, Deirdre O’Brien, and Mike Meyer. Overlapping experiment infrastructure: More, better, faster experimentation. In *Proceedings of the 16th International Conference on Knowledge Discovery and Data Mining*, KDD, pages 17–26, 2010. doi:[10.1145/1835804.1835810](https://doi.org/10.1145/1835804.1835810).
- [287] Silvana Trimi and Jasmina Berbegal-Mirabent. Business model innovation in entrepreneurship. *International Entrepreneurship and Management Journal*, 8(4):449–465, 2012. doi:[10.1007/s11365-012-0234-3](https://doi.org/10.1007/s11365-012-0234-3).
- [288] Orlando Troisi, Gennaro Maione, Mara Grimaldi, and Francesca Loia. Growth hacking: Insights on data-driven decision-making from three firms. *Industrial Marketing Management: the international journal for industrial and high-tech firms.*, 90: 538–557, 2020. doi:[10.1016/j.indmarman.2019.08.005](https://doi.org/10.1016/j.indmarman.2019.08.005).
- [289] Mark Utting, Alexander Pretschner, and Bruno Legeard. A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability*, 22(5):297–312, 2012. doi:[10.1002/stvr.456](https://doi.org/10.1002/stvr.456).
- [290] Arturo Valdivia. Customer lifetime value in mobile games: a note on stylized facts and statistical challenges. In *Proceedings of the 3rd Conference on Games*, CoG, pages 1–5, 2021. doi:[10.1109/CoG52621.2021.9619092](https://doi.org/10.1109/CoG52621.2021.9619092).
- [291] Raymon van Dinter, Bedir Tekinerdogan, and Cagatay Catal. Automation of systematic literature reviews: A systematic literature review. *Information and Software Technology*, 136(106589), 2021. doi:[10.1016/j.infsof.2021.106589](https://doi.org/10.1016/j.infsof.2021.106589).

- [292] Jilles van Gorp, Jan Bosch, and Mikael Svahnberg. On the notion of variability in software product lines. In *Proceedings of the Working Conference on Software Architecture*, WICSA, pages 45–54, 2001. doi:[10.1109/WICSA.2001.948406](https://doi.org/10.1109/WICSA.2001.948406).
- [293] Erno Vanhala and Kari Smolander. What do we know about business models in software companies?: systematic mapping study. *IADIS International Journal on WWW/Internet*, 11(3):89–102, 2013.
- [294] Bruna Prauchner Vargas, Ingrid Signoretti, Maximilian Zorzetti, Sabrina Marczak, and Ricardo Bastos. On the understanding of experimentation usage in light of lean startup in software development context. In *Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering*, EASE, pages 330–335, 2020. doi:[10.1145/3383219.3383257](https://doi.org/10.1145/3383219.3383257).
- [295] Konstantina Vasileiou, Julie Barnett, Susan Thorpe, and Terry Young. Characterising and justifying sample size sufficiency in interview-based studies: systematic analysis of qualitative health research over a 15-year period. *BMC Medical Research Methodology*, 18(1):1–18, 2018. doi:[10.1186/s12874-018-0594-7](https://doi.org/10.1186/s12874-018-0594-7).
- [296] Inder M. Verma. Editorial expression of concern: Experimental evidence of massivescale emotional contagion through social networks. *Proceedings of the National Academy of Sciences of the United States of America (PNAS)*, 111(29):10779, 2014. doi:[10.1073/pnas.1412469111](https://doi.org/10.1073/pnas.1412469111).
- [297] Paul Voigt and Axel von dem Bussche. *The EU general data protection regulation (GDPR): A practical guide*. Springer, 2017.
- [298] Romi Satria Wahono. A systematic literature review of software defect prediction. *Journal of Software Engineering*, 1(1):1–16, 2015.
- [299] Dolores R. Wallace and Roger U. Fujii. Software verification and validation: an overview. *IEEE Software*, 6(3):10–17, 1989. doi:[10.1109/52.28119](https://doi.org/10.1109/52.28119).
- [300] Jason Wang, David Goldberg, Pauline Burke, and Dave Bhoite. Designing and analyzing A/B tests in an online marketplace. In *Proceedings of the 18th International Conference on Data Mining Workshops*, ICDMW, pages 1447–1452, 2018. doi:[10.1109/icdmw.2018.00206](https://doi.org/10.1109/icdmw.2018.00206).
- [301] Qing Wang, Chunqiu Zeng, Wubai Zhou, Tao Li, S. Sitharama Iyengar, Larisa Shwartz, and Genady Ya Grabarnik. Online interactive collaborative filtering using multi-armed bandit with dependent arms. *IEEE Transactions on Knowledge and Data Engineering*, 31(8):1569–1580, 2018. doi:[10.1109/TKDE.2018.2866041](https://doi.org/10.1109/TKDE.2018.2866041).

- [302] Dennis Westermann, Jens Happe, and Roozbeh Farahbod. An experiment specification language for goal-driven, automated performance evaluations. In *Proceedings of the 28th Annual Symposium on Applied Computing, SAC*, pages 1043–1048, 2013. doi:[10.1145/2480362.2480561](https://doi.org/10.1145/2480362.2480561).
- [303] Anna Wiedemann, Manuel Wiesche, Heiko Gewalt, and Helmut Krcmar. Implementing the planning process within DevOps teams to achieve continuous innovation. In *Proceedings of the 52nd Hawaii International Conference on System Sciences, HICSS*, 2019. doi:[10.24251/HICSS.2019.841](https://doi.org/10.24251/HICSS.2019.841).
- [304] Roel Wieringa and Maya Daneva. Six strategies for generalizing software engineering theories. *Science of Computer Programming*, 101:136–152, 2015. doi:[10.1016/j.scico.2014.11.013](https://doi.org/10.1016/j.scico.2014.11.013).
- [305] Roel Wieringa, Neil Maiden, Nancy Mead, and Colette Rolland. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. *Requirements Engineering*, 11(1):102–107, 2005. doi:[10.1007/s00766-005-0021-6](https://doi.org/10.1007/s00766-005-0021-6).
- [306] Roel J. Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. Springer Publishing Company, 2014. doi:[10.1007/978-3-662-43839-8](https://doi.org/10.1007/978-3-662-43839-8).
- [307] Stefan Wiesner, Christian Gorltdt, Mathias Soeken, Klaus-Dieter Thoben, and Rolf Drechsler. Requirements engineering for cyber-physical systems: Challenges in the context of “industrie 4.0”. In *Proceedings of the 2014 International Conference on Advances in Production Management Systems: Innovative and Knowledge-Based Production Management in a Global-Local World*, volume 438 of *APMS*, pages 281–288, 2014. doi:[10.1007/978-3-662-44739-0_35](https://doi.org/10.1007/978-3-662-44739-0_35).
- [308] Ashley Williams. User-centered design, activity-centered design, and goal-directed design: a review of three methods for designing web applications. In *Proceedings of the 27th International Conference on Design of Communication*, pages 1–8, 2009. doi:[10.1145/1621995.1621997](https://doi.org/10.1145/1621995.1621997).
- [309] Claes Wohlin. Second-generation systematic literature studies using snowballing. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering, EASE*, pages 1–6, 2016. doi:[10.1145/2915970.2916006](https://doi.org/10.1145/2915970.2916006).
- [310] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*. Springer Publishing Company, 2012.
- [311] Cara Wrigley and Karla Straker. Designing innovative business models with a framework that promotes experimentation. *Strategy & Leadership*, 44(1), 2016. doi:[10.1108/SL-06-2015-0048](https://doi.org/10.1108/SL-06-2015-0048).

- [312] Huizhi Xie and Juliette Aurisset. Improving the sensitivity of online controlled experiments. In *Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining*, pages 645–654, 2016. doi:[10.1145/2939672.2939733](https://doi.org/10.1145/2939672.2939733).
- [313] Yuxiang Xie, Nanyu Chen, and Xiaolin Shi. False discovery rate controlled heterogeneous treatment effect detection for online controlled experiments. In *Proceedings of the 24th International Conference on Knowledge Discovery and Data Mining*, KDD, pages 876–885, 2018. doi:[10.1145/3219819.3219860](https://doi.org/10.1145/3219819.3219860).
- [314] Ya Xu and Nanyu Chen. Evaluating mobile apps with A/B and quasi A/B tests. In *Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining*, KDD, pages 313–322, 2016. doi:[10.1145/2939672.2939703](https://doi.org/10.1145/2939672.2939703).
- [315] Ya Xu, Nanyu Chen, Addrian Fernandez, Omar Sinno, and Anmol Bhasin. From infrastructure to culture: A/B testing challenges in large scale social networks. In *Proceedings of the 21th International Conference on Knowledge Discovery and Data Mining*, KDD, pages 2227–2236, 2015. doi:[10.1145/2783258.2788602](https://doi.org/10.1145/2783258.2788602).
- [316] Ya Xu, Weitao Duan, and Shaochen Huang. SQR: Balancing speed, quality and risk in online experiments. In *Proceedings of the 24th International Conference on Knowledge Discovery and Data Mining*, KDD, pages 895–904, 2018. doi:[10.1145/3219819.3219875](https://doi.org/10.1145/3219819.3219875).
- [317] Sezin Gizem Yaman, Fabian Fagerholm, Myriam Munezero, Jürgen Münch, Mika Aaltola, Christina Palmu, and Tomi Männistö. Transitioning towards continuous experimentation in a large software product and service development organisation: a case study. In *Proceedings of the 17th International Conference on Product-Focused Software Process Improvement*, PROFES, pages 344–359, 2016. doi:[978-3-319-49094-6_22](https://doi.org/978-3-319-49094-6_22).
- [318] Sezin Gizem Yaman, Tanja Sauvola, Leah Riungu-Kalliosaari, Laura Hokkanen, Pasi Kuvaja, Markku Oivo, and Tomi Männistö. Customer involvement in continuous deployment: A systematic literature review. In *Proceedings of the 22nd International Working Conference on Requirements Engineering: Foundation for Software Quality*, REFSQ, pages 249–265, 2016. doi:[10.1007/978-3-319-30282-9_18](https://doi.org/10.1007/978-3-319-30282-9_18).
- [319] Sezin Gizem Yaman, Fabian Fagerholm, Myriam Munezero, Hanna Maenpaa, and Tomi Mannisto. Notifying and involving users in experimentation: Ethical perceptions of software practitioners. In *Proceedings of the 11th International Symposium on Empirical Software Engineering and Measurement*, ESEM, pages 199–204, 2017. doi:[10.1109/esem.2017.31](https://doi.org/10.1109/esem.2017.31).

- [320] Sezin Gizem Yaman, Myriam Munezero, Jürgen Münch, Fabian Fagerholm, Ossi Syd, Mika Aaltola, Chritsina Palmu, and Tomi Männistö. Introducing continuous experimentation in large software-intensive product and service organisations. *Journal of Systems and Software*, 133:195–211, 2017. doi:[10.1016/j.jss.2017.07.009](https://doi.org/10.1016/j.jss.2017.07.009).
- [321] Sezin Gizem Yaman, Tommi Mikkonen, and Riku Suomela. Continuous experimentation in mobile game development. In *Proceedings of the 44th Euromicro Conference on Software Engineering and Advanced Applications*, SEAA, pages 345–352, 2018. doi:[10.1109/seaa.2018.00063](https://doi.org/10.1109/seaa.2018.00063).
- [322] Scott W. H. Young. Improving library user experience with A/B testing: Principles and process. *Weave: Journal of Library User Experience*, 1(1), 2014. doi:[10.3998/weave.12535642.0001.101](https://doi.org/10.3998/weave.12535642.0001.101).
- [323] Zhenyu Zhao, Miao Chen, Don Matheson, and Maria Stone. Online experimentation diagnosis and troubleshooting beyond AA validation. In *Proceedings of the 3rd International Conference on Data Science and Advanced Analytics*, DSAA, pages 498–507, 2016. doi:[10.1109/dsaa.2016.61](https://doi.org/10.1109/dsaa.2016.61).