ALEKSANDER FABIJAN

DATA-DRIVEN SOFTWARE DEVELOPMENT AT LARGE SCALE

From Ad-Hoc Data Collection to
Trustworthy Experimentation

DATA-DRIVEN SOFTWARE DEVELOPMENT AT LARGE SCALE

Malmö University
Studies in Computer Science No 6,
Doctoral dissertation

© Aleksander Fabijan
ISBN 978-91-7104-918-6 (print)
ISBN 978-91-7104-919-3 (pdf)
Holmbergs, Malmö 2018

ALEKSANDER FABIJAN
DATA-DRIVEN SOFTWARE
DEVELOPMENT AT LARGE
SCALE

From Ad-Hoc Data Collection to Trustworthy
Experimentation

Department of Computer Science
and Media Technology
Faculty of Technology and Society
Malmö University, 2018

Studies in Computer Science

Faculty of Technology and Society
Malmö University

1. Jevinger, Åse. *Toward intelligent goods: characteristics, architectures and applications*, 2014, Doctoral dissertation.
2. Dahlskog, Steve. *Patterns and procedural content generation in digital games: automatic level generation for digital games using game design patterns*, 2016, Doctoral dissertation.
3. Fabijan, Aleksander. *Developing the right features: the role and impact of customer and product data in software product development*, 2016, Licentiate thesis
4. Paraschakis, Dimitris. *Algorithmic and ethical aspects of recommender systems in e-commerce*, 2018, Licentiate thesis
5. Hajinasab, Banafsheh. *A Dynamic Approach to Multi Agent Based Simulation in Urban Transportation Planning*, 2018, Doctoral dissertation
6. Fabijan, Aleksander. *Data-Driven Software Development at Large Scale*, 2018, Doctoral dissertation

Electronically available at:
<https://muep.mau.se/handle/2043/24873>

Dedicated to Annabelle, my daughter.

ABSTRACT

Accurately learning what customers value is critical for the success of every company. Despite the extensive research on identifying customer preferences, only a handful of software companies succeed in becoming truly data-driven at scale. Benefiting from novel approaches such as experimentation in addition to the traditional feedback collection is challenging, yet tremendously impactful when performed correctly.

In this thesis, we explore how software companies evolve from data-collectors with ad-hoc benefits, to trustworthy data-driven decision makers at scale. We base our work on a 3.5-year longitudinal multiple-case study research with companies working in both embedded systems domain (e.g. engineering connected vehicles, surveillance systems, etc.) as well as in the online domain (e.g. developing search engines, mobile applications, etc.).

The contribution of this thesis is three-fold. First, we present how software companies use data to learn from customers. Second, we show how to adopt and evolve controlled experimentation to become more accurate in learning what customers value. Finally, we provide detailed guidelines that can be used by companies to improve their experimentation capabilities.

With our work, we aim to empower software companies to become truly data-driven at scale through trustworthy experimentation. Ultimately this should lead to better software products and services.

Keywords. A/B testing, customer feedback, data-driven development, Experimentation, Experimentation Growth

ACKNOWLEDGMENTS

I would like to thank everyone that have made this work possible and my journey a great experience.

First of all, I would like to thank my main supervisor Helena Holmström Olsson. Her constructive input and always timely feedback were paramount for this work. Helena did an amazing job in bootstrapping my research, in providing valuable input during my PhD years, and in giving me the space to develop my curiosity in the areas that interest me the most. I feel blessed to have Helena as a supervisor, mentor, and as a colleague, and I cannot imagine a journey towards PhD without her.

Second, I would like to thank my co-supervisor, Jan Bosch. Jan's impact on me and my work was twofold. First, and as a director of Software Center, he enabled me to conduct my research in a unique collaboration of practitioners and researchers which were essential for this work. Second, his radical candor (aka scrutinizing my work to a degree that many might not handle) improved my research and the work that I do with orders of magnitude.

Next, I would like to thank everyone at Malmö University that has been involved in the process of this research. Special thanks to Bengt J. Nilsson and Ulrik Eklund for participating in my evaluation group, and to Paul Davidsson as the examiner. In addition to them, I wish to express my gratitude also to Annabella Loconsole, who was responsible to plan my teaching activities and made this process smooth, Romina Spalazzese, whom I learned a lot from during our lunches and breaks, Nancy L. Russo, Thomas Pederson, Anton Tibblin, and everyone else at MAU that was a part of my PhD journey at some point.

Thank you Software Center for funding this research, and to all the companies that have been involved in our projects. Thank you to the main contacts that made this research possible, and the many teams and individuals in them that contributed to our work.

Next, I would like to thank everyone at Microsoft's Analysis & Experimentation team. Special thanks to Ronny Kohavi for enabling me to conduct the internships and a large part of my research in close collaboration with the A&E team. Thank you Pavel Dmitriev for your valuable mentorship during my internships, and during collaborations that emerged as a result of our work together. Special thanks also to Brian Frasca, Somit Gupta and Greg Linden. I have learned so much from everyone in the team during my time at Microsoft!

Finally, I would like to thank Colin McFarland that was previously working with Skyscanner and is now working with Netflix, Lukas Vermeer at Booking.com, and Dylan Lewis at Intuit for our collaborations. Thank you also to other companies and individuals with whom we collaborated throughout this journey.

LIST OF PUBLICATIONS

Included Papers

- 1 A. Fabijan, H. H. Olsson, and J. Bosch, “The Lack of Sharing of Customer Data in Large Software Organizations: Challenges and Implications,” in **Proceedings** of the 17th International Conference on Agile Software Development XP'16, (Edinburgh, Scotland, April 2016), 39–52.

- 2 A. Fabijan, H. H. Olsson, and J. Bosch, “Differentiating feature realization in software product development,” in **Proceedings** of the 18th International Conference on Product-Focused Software Process Improvement, PROFES'17, (Innsbruck, Austria, December 2017), 221–236.

- 3 A. Fabijan, H. H. Olsson, and J. Bosch, “Time to Say ‘Good Bye’: Feature Lifecycle,” in **Proceedings** of the 42th Euromicro Conference on Software Engineering and Advanced Applications, SEAA'16, (Limassol, Cyprus, Aug. 2016), 9–16.

- 4 A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch, “The (Un)Surprising Impact of Online Controlled Experimentation,” in *Revision in an International Software Engineering Journal*.

- 5 A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch, “The Evolution of Continuous Experimentation in Software Product Development,” in **Proceedings** of the 39th International Conference on Software Engineering, ICSE'17, (Buenos Aires, Argentina, May 2017), 770–780.

- 6 A. Fabijan, P. Dmitriev, C. McFarland, L. Vermeer, H. H. Olsson, and J. Bosch, “The Experimentation Growth: Evolving Trustworthy A/B Testing Capabilities in Online Software Companies,” *in Revision in an International Software Engineering Journal*.

- 7 S. Gupta, S. Bhardwaj, P. Dmitriev, U. Lucy, A. Fabijan, and P. Raff, “The Anatomy of a Large-Scale Online Experimentation Platform,” to appear in **Proceedings** 2018 IEEE International Conference on Software Architecture, ICSA'18, (Seattle, USA, May 2018).

- 8 A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch, “Effective Online Experiment Analysis at Large Scale,” in submission, shorter version of this paper to appear in **Proceedings** of the 44th Euromicro Conference on Software Engineering and Advanced Applications., SEAA'18, (Prague, Czechia, August 2018).

- 9 A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch, “The Benefits of Controlled Experimentation at Scale,” in **Proceedings** of the 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications., SEAA'17, (Vienna, Austria, August 2017), 18–26.

- 10 A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch, “Online Controlled Experimentation at Scale: A Survey on the Current State of A/B Testing,” in submission, shorter version of this paper to appear in **Proceedings** of the 44th Euromicro Conference on Software Engineering and Advanced App., SEAA'18, (Prague, Czechia, Aug. 2018).

Personal Contribution

For all of the included papers that I am listed as the first author, I was the main contributor with regard to inception, planning, execution of the research, and writing of the publications. In the single publication included in this thesis and listed as a co-author, I was responsible for inception, data collection, analysis, and writing of one part of the main contributions section, in addition to contributing to the remainder of the paper.

Related Papers Not Included in This Thesis

This thesis work also builds on other related papers that are co-authored by me, which we do not include in this thesis.

A. Fabijan, H. H. Olsson, and J. Bosch, “Data-driven decision-making in product R&D,” in **Proceedings** of the 16th International Conference on Agile Software Development XP’15, 2015.

A. Fabijan, H. H. Olsson, and J. Bosch, “Customer Feedback and Data Collection Techniques in Software R&D: A Literature Review,” in **Proceedings** of Software Business, ICSOB’15, 2015,

A. Fabijan, H. H. Olsson, and J. Bosch, “Early Value Argumentation and Prediction: An Iterative Approach to Quantifying Feature Value,” in **Proceedings** of the 16th International Conference on Product-Focused Software Process Improvement PROFES’15, 2015.

H.H. Olsson, J. Bosch, and A. Fabijan, “Experimentation that matters: A Multi-case Study on the Challenges with A/B Testing,” in **Proceedings** of the Software Business, ICSOB’17, 2017.

A. Fabijan, H. H. Olsson, and J. Bosch, “Commodity Eats Innovation for Breakfast: A Model for Differentiating Feature Realization,” in **Proceedings** of the Product-Focused Software Process Improvement PROFES’16, 2016.

D.I. Mattos, P. Dmitriev, A. Fabijan, H. H. Olsson, and J. Bosch, “Beyond Ad-Hoc Experiments: A Framework for the Online Controlled Experimentation Process,” in **Submission**, 2018.

TABLE OF CONTENT

ABSTRACT	7
ACKNOWLEDGMENTS	9
LIST OF PUBLICATIONS	11
Included Papers	11
Personal Contribution	12
Related Papers Not Included in This Thesis	13
1 INTRODUCTION	23
1.1 Software	23
1.2 Data	25
1.3 Research Objective	26
1.3.1 Research Questions	28
1.3.2 Mapping Of RQs and Chapters	31
1.4 Key contributions of this thesis	32
2 BACKGROUND	33
2.1 Traditional Software Development (TSD)	33
2.1.1 Benefiting from feedback data in TSD	34
2.2 Agile Software Development (ASD)	35
2.2.1 Benefiting from feedback data in ASD	36
2.3 The Continuous Integration and Deployment Step	37
2.3.1 Benefiting from feedback data through CI and CD	38
2.4 The Experimentation Stage	39
2.4.1 Benefiting from data Through Experimentation	40
2.5 Experimentation in Software Product Development	41
2.5.1 Experimentation models	41
2.5.2 Online Controlled Experimentation	43

2.5.3	Why is experimentation so challenging?	45
3	RESEARCH APPROACH	47
3.1	Case Study Research	47
3.2	Case Companies	48
3.2.1	Software Center Companies Collaboration	48
3.2.2	Online Companies Collaboration	52
3.3	Data Collection	53
3.3.1	Joint Company Workshops	53
3.3.2	Individual Company Workshops	53
3.3.3	Reporting Workshops	53
3.3.4	Interviews	54
3.3.5	Internships	55
3.3.6	Tutorials	56
3.3.7	Prototyping	56
3.3.8	Questionnaires	57
3.3.9	Other data collection methods	57
3.4	Data Analysis	59
3.4.1	Qualitative Analysis	59
3.4.2	Quantitative Analysis	61
3.5	Threats to validity	61
3.5.1	Construct Validity	62
3.5.2	Internal Validity	63
3.5.3	External Validity	64
3.5.4	Reliability	64
3.6	Ethical Remarks	65
4	COLLECTING FEEDBACK DATA	67
4.1	Introduction	69
4.2	Background	70
4.2.1	Collection of Customer Feedback	70
4.2.2	Impact and Use of Customer Data	72
4.3	Research Method	72
4.3.1	Data Collection	74
4.3.2	Data Analysis	74
4.3.3	Validity considerations	75
4.4	Findings	75
4.4.1	Data Collection Practices: Current State	76
4.4.2	Data Sharing Practices: Challenges	78

4.4.3	Data Sharing Practices: Implications	80
4.5	Discussion.....	82
4.5.1	The Model: From Quantifiable Feedback to Partial Customer Value	83
4.6	Conclusion.....	86
5	DIFFERENTIATING FEATURE REALIZATION	87
5.1	Introduction	89
5.2	Background	90
5.2.1	Business and Design Experimentation	92
5.2.2	Feature Differentiation	92
5.3	Research Method	94
5.3.1	Data Collection	94
5.3.2	Data Analysis	94
5.3.3	Threats to Validity	95
5.4	Empirical Findings.....	96
5.4.1	Current State of Feature Differentiation	97
5.4.2	Challenges with Feature Differentiation.....	98
5.4.3	Implications	99
5.4.4	Summary of our empirical findings.....	100
5.5	Differentiating Feature Realization	101
5.5.1	Feature Differentiation	102
5.5.2	Activity Prioritization	103
5.6	Model Evaluation	105
5.7	Output-Driven vs. Outcome-Driven Development	107
5.7.1	Output-Driven Development.....	108
5.7.2	Outcome-Driven Development.....	109
5.8	Conclusions	110
6	FEATURE LIFECYCLE.....	111
6.1	Introduction	113
6.2	Background	114
6.2.1	Product Lifecycle	114
6.2.2	Feature Value	116
6.3	Research Method	117
6.3.1	Data Collection	118
6.3.2	Data Analysis	119
6.3.3	Validity considerations	119
6.4	Findings	120

6.4.1	Current State	120
6.4.2	Challenges	122
6.5	Feature Lifecycle Model	123
6.5.1	Feature Value Equation	124
6.6	Feature Lifecycle Model	126
6.7	Discussion.....	131
6.8	Conclusions and Future Work	133
7	EXPERIMENT LIFECYCLE	135
7.1	Data-Driven Decision-Making	137
7.2	Online Controlled Experiments	137
7.3	The Impact of Experimentation	138
7.4	Experiment Lifecycle	139
7.4.1	Ideation.....	140
7.4.2	Design & Execution	140
7.4.3	Analysis & Learning	141
7.4.4	Challenges & Mitigations.....	141
7.4.5	Example Experiments	143
7.5	Integration Yields Impact.....	147
7.6	Conclusion.....	148
8	EVOLVING EXPERIMENTATION	149
8.1	Introduction	151
8.2	Background	153
8.2.1	Learning what customers Value	153
8.2.2	From Data to Valuable Insights	154
8.2.3	Online Controlled Experimentation.....	156
8.3	Research Methodology	158
8.3.1	Research Process	158
8.3.2	Phase 1 – In-Depth Single-Case Study at Microsoft	159
8.3.3	Phase 2 – Multiple-Case Study.....	160
8.3.4	Discussion on the Validity.....	163
8.4	Empirical Data	164
8.4.1	L1: Data Quality Fosters Trustworthiness	165
8.4.2	L2: Platform Accelerates Experimentation Growth	167
8.4.3	L3: Qualitative Insights Enrich Experiment Learnings	169

8.4.4	L4: Iterating OCEs with Small Changes Yields	
	Large ROI	172
8.4.5	L5: The Impact of Experimentation is Boundless	175
8.5	The Experimentation Growth Model	179
8.5.1	Prerequisites for Experimentation	179
8.5.2	Experimentation Growth	183
8.5.3	Applying The Experimentation Growth Model	200
8.5.4	The Challenging Side of Experimentation	203
8.6	Conclusion	205
9	EXPERIMENTATION PLATFORM	207
9.1	INTRODUCTION	209
9.2	BACKGROUND & RESEARCH METHOD	212
9.2.1	Online Controlled Experiments	212
9.2.2	Related Work	213
9.2.3	Research Method	213
9.3	ExP PLATFORM ARCHITECTURE	215
9.3.1	Experimentation Lifecycle	215
9.3.2	Experimentation Portal	216
9.3.3	Experiment Execution Service	222
9.3.4	Log Processing Service	226
9.3.5	Experiment Analysis	229
9.4	CONCLUSION	234
10	EFFECTIVE EXPERIMENT ANALYSIS	235
10.1	Introduction	237
10.2	Background	238
10.2.1	Online Controlled Experiments	239
10.2.2	Experimentation Metrics	239
10.2.3	Experimental Design	240
10.3	Research Method	241
10.3.1	Data Collection	241
10.3.2	Data Analysis	243
10.4	Analyzing Online Controlled Experiments	243
10.4.1	Standard Scorecard Analysis	243
10.4.2	Guided Experiment Analysis	248
10.5	Evaluation & Discusion	252
10.5.1	Evaluating Guided Experiment Analysis	252
10.5.2	Evaluation Results	254

10.6	Threats to validity & Limitations	256
10.7	Conclusion.....	258
11	THE BENEFITS OF EXPERIMENTATION	259
11.1	Introduction	261
11.2	Background	262
11.2.1	Online Controlled Experimentation	263
11.2.2	Metrics and Value.....	264
11.2.3	Leading and Lagging Indicators	265
11.3	Research Method	265
11.3.1	Data Collection	266
11.3.2	Data Analysis	267
11.3.3	Validity Considerations	268
11.4	Empirical Data and Findings	269
11.4.1	Office Contextual Command Bar Experiment	269
11.4.2	OneNote Publishing Pane Experiment	270
11.4.3	Bing Search Results Birthday Greeting Experiment	273
11.4.4	Bing Background Color Experiments.....	274
11.4.5	Skype Infrastructure Migration	274
11.5	The Benefits of Controlled Experimentation	275
11.5.1	Portfolio Level	275
11.5.2	Product Level	277
11.5.3	Team Level	279
11.6	Challenges with CE	281
11.7	Conclusions and Future Work	283
12	THE CURRENT STATE OF EXPERIMENTATION	
MATURITY		285
12.1	Introduction	287
12.2	Background	289
12.2.1	Learning From Data	289
12.2.2	Online Controlled Experiments.....	290
12.2.3	The Experimentation Growth Model	291
12.3	Research Method	293
12.3.1	Experimentation Growth Survey	293
12.3.2	Data Collection	294
12.3.3	Data Analysis	294
12.3.4	Threats To Validity	295

12.4	Survey Findings & Implications	296
12.4.1	Product Organizations	296
12.4.2	Experimentation Maturity.....	297
12.4.3	Key Findings and Challenges	307
12.5	Conclusion and Future Work.....	309
13	DISCUSSION	311
13.1	Research Questions and Contributions	312
13.1.1	RQA1: How do software companies collect customer feedback and product data?	312
13.1.2	RQA2: How can software companies use the feedback data that they collect to effectively realize software features?	314
13.1.3	RQB1: How can software companies adopt and evolve experimentation?	317
13.1.4	RQB2: How can software companies improve their experimentation capabilities?.....	321
13.1.5	RQB3: How evolved are the experimentation capabilities of software companies, and what are the benefits of becoming data-driven at scale?.....	326
13.2	From Data to Data-Driven Companies.....	328
13.2.1	Accuracy	328
13.2.2	Iterative and Reproducible	329
13.2.3	Completeness	330
13.2.4	Continuous Improvement	331
14	CONCLUSION	333
14.1	Future work.....	334
	BIBLIOGRAPHY	337

1 INTRODUCTION

“Our success at Amazon is a function of how many experiments we do per year, per month, per week, per day...”

-- Jeff Bezos, CEO @ Amazon [1]

1.1 Software

Software functionality was for decades predominately required in products to support the electrical, hardware or mechanical solutions without delivering any other perceptible value [2], [3]. Companies developed software as a necessary cost without exploring the value of software as such. Today, however, systems that were once built purely from hardware components such as e.g. connected cars and household appliances, nowadays contain software functionality that enables them to connect to the Internet, exchange information with other systems, adapt to our needs, and self-improve over time. Software in most products today enables individuals and businesses around the world to achieve more than what has ever been possible before. Swiftly, software is becoming the main competitive advantage in systems that previously existed without it [2], [4]–[6].

Furthermore, and with advances in software development such as Continuous Integration [7] and Continuous Deployment [8], adding new software capability to products has become easier than ever before. Consider for example the applications installed on your cell-phone or in your new car. Every few days new features are added to them, promising the many benefits for improving our

daily lives. Unfortunately, too often the value that new or updated features promise to deliver is merely a proposition that has not been accurately validated. One of the reasons for this is the lack of capability in software companies to systematically and continuously evaluate the value of the changes that they do to their products [9], [10]. Consider, for example, in how many different ways the applications that you use every day could be improved to better meet ‘your’ needs. The possibilities are endless, and some are more costly and less valuable than others [6]. The line between valuable software functionality and wasteful features is thin, yet being on the right side of it is paramount for the success of software companies [11].

Thankfully to modern development approaches such as Agile Development [12] and technological advances such as Continuous Deployment, moving from one side of the line to the other is possible. For example, updating a sub-optimal ranking algorithm on Bing.com and make the update available to hundreds of millions of users around the world can be done in minutes [13]. Drawing the line to determine what is valuable, however, is difficult [14]–[16]. Accurately determining on which side of the line new changes are, is even more challenging [17]. Consider for example an improvement to the product done by changing the way how an Artificial Intelligence (A.I.) algorithm behaves. Determining the success of such changes in the world where heuristics are increasingly more infused in the essential functionality of the product requires a continuous evaluation of product value through the data that is generated by the use of the products. Using the data to experiment with different variants and configurations is paramount to discover what delivers value to users [11].

1.2 Data

In the same way as software became the ‘icing on the cake’ in most products and services, so has the data that is generated through its use [18]–[20]. At first, the data that software products created was used for debugging and understanding when a product crashed. Developers would analyze the output from trace statements or crashes in source code and use it to repair the product [21], [22]. Today, software logging practices have evolved in the orders of magnitude [22], [23]. Product logs now give data-driven companies the possibility to learn how their new features or product changes are accepted, and how much value they provide (if any at all). The use of data that describes how the product is **actually being used** is becoming a standard operating procedure in data-driven companies while prioritizing development decisions [11], [24].

Large-scale companies (e.g. producers of products used by thousands, millions or billions of users) depend on learning from data being generated through the use of their products. Knowing what does and what does not deliver value empowers software companies to accurately prioritize the items on the product roadmap, ultimately leading to better products and services [11], [25]. Consider for example Amazon, Booking.com or Microsoft, where most changes to their products, from the smallest bug fixes to the most impactful new feature, are deployed to their customers in a controlled and scientific process, guaranteeing that those that received the new updates really experience the expected benefits [11], [25]. Or, more common, to discover benefits that the companies did not know exist. Through experimentation and fast feedback loops, data-driven companies are transforming into experimental laboratories, learning faster than their competitors and steadily growing their businesses [11].

For many companies, accurately learning and benefiting from the data is challenging. First, the collection of the data needs to be implemented. The list of possibilities to learn from customers is long, the list of pitfalls and caveats while doing this even longer [16], [26]. Next, infrastructure to process and analyze the data

needs to be set-up. Moreover, technical challenges are not the only setback [27]. Consider for example the culture in companies that have for decades celebrated a new version or a new release of their product. At Microsoft, the practitioners received a “ship-it award” for successfully deploying their new features. Today, the reward is given when measurable value is confirmed. And as companies increasingly shift towards services where revenue income is conditional on customer retention and continuing use of the product, learning from customers with scientific accuracy is critical.

In this thesis, we explore how companies collect and use customer feedback and product data to better prioritize product development activities for the long-term value of the products that they develop. We investigate how companies experiment with their products while they are in use, and we construct an understanding of how the experimentation capabilities can be improved to become more effective. Ultimately, we show how to transform the software development process into an experimental laboratory, enabling practitioners to conduct decisions with scientific accuracy.

1.3 Research Objective

As described earlier, companies are successful only if they truly understand customers’ needs, and develop products or services that deliver value to fulfil them [6], [28]–[31]. Most software project failures are caused by value-oriented shortfalls such as the lack of customer or user feedback [32]–[35] that cause companies to deliver features that do not add value..

Learning from and about customers and their needs is the essential unit of progress for companies whether developing a new product or improving an existing feature. Unfortunately, **efficiently** and **accurately** learning from customer feedback and product data is challenging [10], [15], [36], [37]. To understand what the customers say they value and make decisions based on the lessons learnt, companies have always been collecting feedback [38]–[41]. For example, interviews, surveys, observations etc. are techniques that are regularly used to involve with the customers and collect feedback. On top of these typically very qualitative techniques, and

with the capability to connect software products to the internet, new opportunities are emerging to learn from customers while they use of the product. Today more data can be generated than ever before to learn about customer habits, needs and preferences [20], [42], [43]. For example, telemetry data in connected cars can reveal in which way a vehicle is used and recommend maintenance activities, increasing the lifespan of the product. On the web, clicks on search results provide feedback to the ranking algorithm, improving the search algorithm for every next query, generating new hypotheses on how the product team can improve the system[44]. And as software products move away from large periodic releases (e.g. yearly versions) and closer towards continuous deployment of new functionality (e.g. daily builds), accurately evaluating what delivers value to the customers is of the paramount importance.

The main objective of this thesis is twofold. First, we aim to explore how large-scale companies (companies that develop software products used by thousands, millions or billions of users) collect and use customer feedback and product data to become data-driven. Our second objective is to provide detailed guidelines on how to become more accurate in this by adopting and evolving controlled experimentation. In our view, and as we discuss in chapter 13.2, Data-Driven Companies can be seen as experimental laboratories that employ scientific trustworthy procedures to improve their software products and data handling procedures. Investigating how to become data-driven through experimentation is thus of great interest for many companies.

1.3.1 Research Questions

1.3.1.1 *Data-driven development*

As demonstrated in prior research, companies today have the possibility to learn from large amounts of data [45], [46]. Ideally, companies should be able to utilize the feedback data that they collect to accurately learn what products and features to develop, to what extent, and when to stop developing them. Data should drive discussions, improve the company, and be used to decide when and which teams to reward. Unfortunately, accurately learning from data is very challenging. The alternative is to use opinions and intuition, however, the intuition of software development companies on customer preferences can be as much as 90% of the time wrong [32], [47], [48]. Software products are frequently overloaded with features that rarely even get used by customers [32].

In order to effectively develop software products, and to help companies move away from assumptions that could be harmful, more knowledge is needed in how to use and collect feedback. For the purpose of this work, we define feedback as data that companies actively solicit or passively receive from users of their products or products directly, with the intent to understand what features or changes to the product deliver value to users. This led us to our first research question:

RQ_A: “How do large scale data-driven software companies learn from their customers to effectively develop software products that deliver long-term value?”

We acknowledge that in some context ‘customer’ and ‘user’ can mean refer to different stakeholders. For example, a customer could be the entity purchasing the product which is then used by the users. However, in our research we use the terms interchangeably to refer to the users of the product unless explicitly stated otherwise.

To address our first research question, we divided it into two related sub-questions, each addressing one of the dimensions of effective data-driven development:

- **RQ_{A1}** : “: How do software companies **collect** customer feedback and product data?
- **RQ_{A2}** : “: How can software companies use the feedback data that they collect to **effectively realize software features**?

We answer RQ_{A1} by recognizing feedback collection techniques that are in use by large-scale software companies. We answer RQ_{A2} by designing a model for feature differentiation and feature value evaluation in software product development.

1.3.1.2 *Data-Driven development through Experimentation*

At the time of this research, advances in Continuous Integration [7], [49], [50] and Continuous Deployment [51]–[54] enabled companies to quickly build enhanced versions and variants of their products, and deploy them to their customers over the Internet in several different domains. The notion of **experimentation** emerged as the natural next step in the chain of the software release steps [52], [55] by enabling companies to make causal conclusions between the changes introduced to their products (e.g. new features) and customer reactions on them (e.g. using the feature more or less frequently).

Several large-scale online companies such as Google, Microsoft and Amazon previously reported on the benefits that they experience due to the adoption of the experimentation practices in their companies [25], [47], [56]–[58]. However, the reminder of the software industry continues to struggle to adopt and evolve their experimentation capabilities [4], [10], [36], [59]. In addition, the experimentation maturity of the companies that already experiment is unclear, leaving us with vague understanding of experimentation and its adoption in the software industry. The aforementioned challenges motivated our next research question.

RQ_B : “: How can large-scale software companies improve their data-driven capabilities through experimentation?

To address this high-level research question, we divided it into three sub-questions, each addressing one of the aspects of data-driven development through experimentation:

- **RQ_{B1} :** “: How can software companies **adopt and evolve experimentation**?
- **RQ_{B2} :** How can software **companies improve their experimentation capabilities**?
- **RQ_{B3} :** How **evolved** are the experimentation capabilities of software companies, and **what are the benefits** of becoming data-driven at scale?

We answer **RQ_{B1}** by introducing the Experiment Lifecycle and the Experimentation Growth Model in chapters 7 and 8. We answer **RQ_{B2}** by providing an architecture of a large-scale experimentation system and a tool that improves the experiment analysis effectiveness in chapters 9 and 10. Finally, we answer **RQ_{B3}** in chapter 11 by providing an overview of the current state of experimentation maturity among companies that already experiment, and demonstrating what the benefits of this method are in chapter 12.

1.3.2 Mapping Of RQs and Chapters

In this section we briefly summarize our research questions and map them to the following chapters of this thesis.

HRQ	Research Question	CH
<i>RQ_A</i>	<i>RQ_{A1}</i> : How do software companies collect customer feedback and product data?	4
	<i>RQ_{A2}</i> : How can software companies use the feedback data that they collect to effectively realize software features?	5, 6
<i>RQ_B</i>	<i>RQ_{B1}</i> : How can software companies adopt and evolve experimentation?	7, 8
	<i>RQ_{B2}</i> : How can software companies improve their experimentation capabilities?	9, 10
	<i>RQ_{B3}</i> : How evolved are the experimentation capabilities of software companies, and what are the benefits of becoming data-driven at scale?	11, 12

In the next section, we summarize the key contributions.

1.4 Key contributions of this thesis

The key contributions of this thesis are the following:

1. Improved scientific knowledge about the collection of customer feedback and its use.
 - We provide a state-of-the-art review of techniques for the collection of customer feedback and product data.
 - We present a framework for feedback use in feature differentiation and reasoning about long-term feature value.
2. Novel scientific knowledge of experimentation and its adoption in the software industry.
 - We present the Experimentation Growth Model (EGM), introducing the seven most critical dimensions of adopting and evolving experimentation.
 - We present the current state of Experimentation Growth based on the EGM.
3. We provide in-depth guidance for improving three dimensions of the Experimentation Growth Model.
 - We provide a detailed architecture of an experimentation system, addressing the first dimension of the EGM.
 - We present and discuss the analysis of experiments at large scale, and develop and evaluate an artefact that improves the efficiency and the comprehensiveness of the analysis, addressing the fourth dimension of the EGM.
 - We present and discuss the impact and implications of experimentation at large scale, addressing the seventh dimension of EGM.

2 BACKGROUND

In the previous chapter, we introduced our research objectives. In this section, we present the background and the context of the problem domain. To structure the section, we follow the steps described in the “Stairway to Heaven” (StH) model [52]. StH model describes an evolutionary process of evolving from companies with traditional software development process towards an experimentation-focused process. We conclude the section with an emphasize on the final stage of the model – the experimentation stage – within which our research is being conducted.

2.1 Traditional Software Development (TSD)

The first step of the StH model describes Traditional Software Development (TSD) [60]. TSD, also called waterfall or serial approach towards developing software, is based on the idea to organize the development of a software product into phases or stages. The stages of TSD span from the requirements stage, to architecture, design, development, test, and finally towards the deployment of the product. Models that organize development following TSD have been used to manage the development of large projects for many years, and remain in some companies still in use (e.g. the V-model [61] in the automotive industry, PRINCE2, etc.).

In traditional waterfall development model and its improvements (e.g. the double-loop model introduced by Royce [60]) there is a large emphasize on learning from customers in the early stages. Companies using TSD should heavily invest into studying custom-

ers to determine, as accurately as they can, what to develop before starting the implementation. Therefore, the topic of customer and user feedback has been increasingly studied in Requirements Engineering domain [62]–[66], Information Systems [67], [68], Human-Computer Interaction [69], [70], Product Management [40], [71], and Entrepreneurship [31], [72].

Although very valuable in projects where requirements do not evolve, TSD approaches are subjects to many challenges and critiques. One significant challenge with TSD approaches is that due to the waterfall-style interactions, typically a substantial amount of time can pass between the time that requirements were collected and until the product was used by the customers for the first time [60]. This has implications to the way in which companies can benefit from the feedback and data that they collect, which we discuss next.

2.1.1 Benefiting from feedback data in TSD

In TSD and as described above, the first stage – Requirements – focuses on eliciting and collecting feedback. Asking customers about their needs and preferences has always been one of the most fundamental parts of any product development process [38]–[41], [73], [74]. Especially in those processes that are based on TSD. Examples of techniques for collecting feedback to create requirements on what to build are e.g. interviewing [40], [75], [76] conducting user studies [76], [77], experience querying [78], [79], usability studying [80], filming and conducting observations [75], [40], storyboarding [81], voting [82]–[84] etc.

The strength of these collection techniques is their ability to provide rich typically textual descriptions, opinions, and information of how individuals experience a given situation [85]. The typically qualitative data collected through these techniques helps in creating requirements on how to design the product and what needs to be developed. As such, techniques collecting qualitative feedback provide in-depth information about the ‘human’ side of the product usage and their needs [85]. Another way in which such data can be used is illustrated by Kano [86], who recognized the need to use

the feedback to differentiate between product attributes (e.g. attractiveness, satisfaction, must-be, indifferent and reverse attributes) based on their importance for the product and the stakeholders.

However, while qualitative feedback and requirements generated in this way often contain information about how much users like or dislike new features [59], they can be highly biased or incomplete. For example, users who disliked the feature are generally more likely to provide feedback than those that liked it [87]. Therefore, this type of feedback frequently does not provide an objective way to evaluate the overall impact of change in the software product. And in the case of Traditional Development Approaches, the feedback is infrequent and often too late [50], [88].

The issues with late feedback and slow deployment cycles were partially mitigated with the introduction of the Agile approaches towards software development, which we briefly describe next.

2.2 Agile Software Development (ASD)

“Plans are worthless, but planning is everything”

-- Dwight D. Eisenhower

One of the main reasons why TSD approaches do not suffice in rapidly changing business environment is that requirements evolve quickly and become obsolete even before software product can be completed or delivered to the customer [89], [90]. The second step of StH model pinpoints Agile Software Development (ASD) as a response for business environments with high risk of change [75].

In Agile development rapid delivery of an application in complete functional components is emphasized. Instead of creating large tasks and schedules, work is done in “sprints.” Each sprint has a defined duration (usually a few weeks) with a running list of deliverables, planned at the start of the sprint. Deliverables are prioritized by business value as determined by skid communicating with the customer. ASD approaches reduce the cost of change throughout a development project by introducing short cycles be-

tween planning, development and the deployment of the product prototypes. ASD approaches such as Extreme Programming, Scrum, and other similar methodologies such as Kanban [91], were adopted by many companies worldwide [92].

To improve the ASD process even further, the concept of Lean thinking that originates from the Toyota Production System [93] was adopted in some organizations. The basic idea of the Lean concept is to optimize the development activities by identifying and removing process constraints [94] e.g. by bringing the testing of the software products closer to the developers), minimizing wasteful activities [95], [96] etc.

The developments in this era enabled new possibilities for learning from customers and users of software products, which we describe next.

2.2.1 Benefiting from feedback data in ASD

In Agile development, the feedback is distributed across the complete development of the product and different types of risks can be easier identified and managed [97]. For example, paper or mockup prototyping [98]–[101] has always been useful to better understand customer needs. However, in ASD and by delivering smaller chunks of working product incrementally to the customers, companies could truly start to develop prototypes that can be used by customers and detect discrepancies that could be missed otherwise [76], [99], [102], [103].

ASD overcomes the fallacy of TSD with collaboration, communication, incremental progress and continuous improvement. The emphasize and value of ASD methods is development-centered as they focus on how to efficiently develop new features [104], [105]. For example, ASD methods allow reprioritizing which features to develop during the development process by reordering the items in the product backlog based on the feedback received by asking the customers about the current prototype / version [10], [106]. One critical aspect of ASD are frequent releases of new code, which became a reality with the introduction of Continuous Integration and Continuous Deployment.

2.3 The Continuous Integration and Deployment Step

Building on the opportunity of Agile methodologies, Continuous Integration (CI) and Continuous Deployment (CD) have had a big impact on the way in which software products are being developed today[107]. Continuous Integration is a software development practice where software engineers integrate their work frequently into a central repository. Usually each person integrates at least daily - leading to multiple integrations per day [7]. Each integration is built and tested to detect integration errors as quickly as possible.

Continuous deployment is the next step forward where software builds that were successful and passed the integration tests get deployed to the production systems [51], [108]. In a scenario where production systems also adopt the new code, the process is commonly named Continuous Delivery [108], [109]. In this thesis, we do not differentiate between Deployment and Delivery, however, we acknowledge that in some scenarios delivering new software updates can be more challenging as deploying them.

CI and CD can be considered as one of the key components of Development and Operations (DevOps) paradigm, which changed the way in which software products and services are developed [54], [88], [108]. Thanks to DevOps, disconnects between the people developing a product and teams supporting its operation that exist in approaches prior to DevOps (see e.g. ITIL [110]) were mitigated through reorganization of work and ownership [51], [111], [112].

In addition to many other benefits such as providing the possibility to realize frequent releases that are essential for ASD, CI and CD help companies become more data-driven. Through the continuous exchange of information, practitioners can collect measurements from the product while being developed and used by customers. This opens new possibilities of collecting and benefiting from data.

2.3.1 Benefiting from feedback data through CI and CD

As opposed to TSD and ASD alone where the feedback is typically elicited and collected by communicating with the customer or user, CI and CD enabled the collection of data directly from the product.

Through the bidirectional channel established for the deployment of new code (e.g. the next release with new features), data about the use of the product can also be collected. For example, companies can collect static attributes (e.g. demographics information), operational data (e.g. system load, performance insights, crashes, etc.) and product usage data (e.g. clicks, dwell times, etc.) [21], [33].

Companies can learn from the product usage data that they collect in several different ways [113]. One possibility is through exploratory data analysis, which helps understand the data when little or no hypotheses exist, or when specific hypotheses exist but supplemental representations are needed to ensure the interpretability of the results [114]. For example, explorative data analysis can help in identifying, among others, customer demographics that use a certain feature more or less, how often they use the product, what version of the product is most popular in a certain segment of customers, and finding other useful insights [45], [114], [115]. Overall, the goal of exploratory data analysis in product development is to identify insights that may lead to hypotheses about the products and customers using them [40], [116].

To understand the customer behavior and discover relationships between multiple variables (for example, the version of a product and a customer segment), analysts can organize the data at hand in pairs (or even higher dimensions), and evaluate whether the evaluated variables are correlated. To evaluate relationships, analysts can use common measures such as Person correlation coefficient, multiple regression, or other statistical techniques [117].

An important limitation of correlational designs, however, is that they cannot be used to draw conclusions about the causal relationships among the measured variables [118]. For example, if practitioners were interested in knowing whether the impact that

was observed by introducing a new software is due to the feature, solely correlation cannot be used to infer this. The next step in the ‘stairway to heaven’ model is addressing this limitation, and aiming to bridge the gap between the development and operations organization through methods where causal inference can be detected – experimentation.

2.4 The Experimentation Stage

The final step on the “stairway to heaven” is where the R&D organization responds and acts based on instant customer feedback, and where actual deployment of software functionality is seen as a way of experimenting. Companies that have adopted agile software development practices [90], and introduced continuous integration and deployment [108], [119], generally produce higher frequencies of software releases which are also smaller in size, compared to those following traditional software development methodologies [107]. This ability, to deploy and revert small changes to and from software products with high velocity, empowers software companies to start experimenting with the products that they develop while they are in the hands of their users [30], [46]. The goal of such experiments is to continuously identify what delivers business value to customers. Fitzgerald [88] coined the term BizDev to describe this process of aligning business strategy and development efforts, and stressing that the link needs to be continuous. Davenport [55] on the other hand, recognized that if a company does a substantial amount of prototyping by running frequent experiments, it will increase the ability to correctly decide which initiatives to support in the future. Moreover, experimentation can provide companies with the ability to discover new business models [11], and empower individuals in organization to become intrapreneurial – turning ideas into value creating services and products [120].

2.4.1 Benefiting from data Through Experimentation

In a typical experiment discussed in this thesis, quantitative data on product usage (clicks, dwell times, time spent, etc.) are collected and used to compute metrics [9], [121]. Most commonly, metric data is computed for every variant in order to determine whether they differ with statistical significance. Metrics are then used to decide on future product development, to set team goals, decide which features should be released to customers, and how resources should be allocated [44], [122].

As opposed to other techniques where data can be used to find interesting insights about customer habits or correlational connections, experimentation can be used to answer causal questions.

We summarize the software engineering advancements presented in this chapter along with some of their benefits and challenges (for a detailed descriptions see e.g. [52], [123]) on an adapted version of the Stairway to heaven model on Figure 1 below.

Figure 1. Focus, Benefits and Challenges of the advances based on an adapted Stairway to Heaven Model.

In the remainder of this chapter, we discuss the final step – Experimentation – in greater detail.

2.5 Experimentation in Software Product Development

Experimentation has been established in software engineering research for many years (see for example Basili et al. [124], Wohlin et al. [125], Juristo et al. [126]). Systematic experimentation in practice, however, is relatively new and underexplored area. We start by introducing models for experimentation that were presented in prior research, and introducing experimentation techniques. Next, we present one of this experimentation techniques - online controlled experimentation in greater detail. We conclude the section by discussing why is experimentation so challenging.

2.5.1 Experimentation models

There are a number of different models that embed the idea of experimentation with software features in software product development. In this section, we briefly introduce four most relevant to the work in this thesis.

2.5.1.1 Hypex Model.

To our knowledge, the Hypothesis Experiment Data-Driven Development (HYPEX) model [30] is one of the first attempts to support companies in initiating, conducting and evaluating feature experiments, focusing on shortening the feedback loops during the development of a feature. The Hypex model builds on the ‘Build-Measure-Learn’ methodology presented by Ries [31] and Blank [127]. In this model, Olsson et al. suggest to deploy a minimal viable instance of the functionality under development to customers and collect relevant quantitative feedback through metrics (e.g. system performance as % of CPU utilization). Next, collected data should be compared to previously defined and expected behavior and any difference examined. In this way, practitioners can confirm whether the feature that they developed delivered the expected

(e.g. decrease of CPU utilization), or if any other work such as further feature improvement needs to be performed.

2.5.1.2 *QCD model*

The Hypex model introduced in the previous section favors quantitative data collection and comparison for hypothesis testing. To contrast this, Olson et. al introduced the QCD model [46] as a response to the needs to integrate qualitative customer feedback in the experimentation process. The QCD model combines different types of feedback data and leverages the strength of synthesis and triangulation. As opposed to the Hypex model, it enables the practitioners to test their hypothesis also based on qualitative data, and to derive future feature development. Ultimately, it can help companies move away from early specification of requirements and towards continuous validation of hypotheses.

2.5.1.3 *The RIGHT model*

In addition to Hypex and QCD models, Fagerholm et al. [4], [49] created a model for continuous experimentation with software-intensive products that consists of building blocks. Their model - Rapid Iterative value creation Gained through High-frequency Testing (RIGHT) - consists of repeated Build- Measure-Learn blocks. These blocks are supported by infrastructure (e.g. Experiment database, analysis tools, etc.). The model connects requirements, design, implementation, testing, deployment, and maintenance phases of software development, and provides a general description of an experimentation process and needed infrastructure.

2.5.1.4 *The Continuous* framework*

The Hypex, QCD and the RIGHT models focus primarily on the Research & Development part of software development. In contrast, the Continuous* framework introduced by Fitzgerald et al. [111] starts by introducing a need for a closer integration of the Business Strategy with the R&D. The Continuous* framework consists of a loop between Business Strategy, Development and Operations, highlighting the importance of information exchange across the organization. The close integration between Development and Operation (DevOps) and Business Strategy and Development (BizDev) is critical to enable, among others, continuous planning, budgeting, verification evaluation and improvement of software projects. The foundation of the framework, however, are continuous innovation and continuous experimentation.

2.5.2 Online Controlled Experimentation

In software product development, the term Experimentation can be used to describe different techniques for exploring the value of the changes introduced to a product [128]. For example, experimentation could refer to iterations with prototypes in the startup domain [31], [127], canary flying [119] of software features (exposing a small percentage of users to a new feature or a change), gradual rollout [119] (deploying a change to one customer groups and expanding to the next one), dark launches [8] (releasing new features disabled and testing them in production), and controlled experimentation [129] (releasing multiple variants of the product and evaluate the differences between them through statistical tests).

In this thesis, and when we discuss experimentation, we refer to controlled experimentation. Other labels that are frequently used to describe controlled experimentation are also A/B testing, split testing, bucket testing and randomized trials.

Controlled experimentation is a scientific method that has been used for decades in the pharmaceutical industry to test the effectiveness of new drugs [16], [130]. The theory of controlled experiments dates to Sir Ronald A. Fisher's experiments at the Rothamsted Agricultural Experimental Station in England during the 1920s [36]. In the simplest controlled experiment, two comparable groups (hence the name A/B) are created by randomly assigning experiment participants in either of them; the control and the treatment. The only thing different between the two groups is some change, e.g. a different design solutions, or the variants communicate with a different back-end server. Through statistical testing, experimentation enables high accuracy of learning from customers that use the product [14]. External factors such as seasonality, impact of other product changes, competitor moves, etc. are distributed evenly between control and treatment variants, and therefore do not impact the results.

In a correctly performed controlled experiment, any difference in metrics between the two groups must be due to the change X or a random chance. The latter is ruled out using statistical testing through e.g. a student's t-test or a chi-square test (the selection of the test depends, among others, on the metric design [17]). The controlled experiment design establishes a causal relationship between the change X made to the product and changes in user behavior measured through metrics. On top of the benefits of CI and CD, the high accuracy of learning can be done very frequently, enabling companies to be both fast and precise in evaluating how their changes to products satisfy their customers.

Experimentation, however, is only gaining momentum across the software industry [10], [131]. In the next section, we discuss briefly its adoption and underlying challenges.

2.5.3 Why is experimentation so challenging?

Conducting trustworthy controlled experiments at large-scale is hard. Among others, establishing experimentation requires commitment from multiple roles in a company, and tremendous investment into experimentation infrastructure, education, and cultural evolvement [13], [16], [132]. Many companies, including major digital enterprises and smaller startups, are still haphazard in their experimentation approach, don't know how to run rigorous scientific tests, or conduct way too few of them [10], [11]. We illustrate why running experiments is so challenging with an example of a **single** isolated experiment.

Consider the following experiment that ran at one of our case companies which we introduce in Chapter 3.2. In this experiment, when the user of the app in which the experiment was run attempts a call but the caller does not answer, a dialog is shown prompting to leave a video message (as opposed to a control variant where such a message would not appear). Many steps need to happen to execute an experiment like this [13].

First, the code that shows the message needs to be deployed to the clients in such a way that it can be turned on and off via the experimentation system. Next, the audience, the experiment design, the experiment steps, and the size and duration of each step need to be determined.

During the experiment execution, correct configurations need to be delivered to the users' mobile and desktop apps, ideally in a gradual manner, while verifying that the experiment is not causing unintended harm (e.g. app crashes). Such monitoring should continue throughout the experiment, checking for a variety of issues, including interactions with other concurrently running experiments. Various actions could be suggested to the experiment owner (the person who executes the experiment), such as stopping the experiment if harm is detected, looking at surprising metric movements, or examining a specific user segment that behaves differently from others.

After the experiment, results need to be accurately analyzed to make a ship/no-ship recommendation and learn about the impact of the feature on users and business. The outcome of the experiment should not only impact the current experiment but also to inform future experiments.

Furthermore, systematically conducting hundreds or thousands of experiments like the one above and effectively learning from them is even more challenging [13], [14]. Companies that take on the challenge of introducing experimentation in their software development struggle extensively and do not advance at a pace that they aim for [10], [27]. First, there are many technical challenges such as instrumenting products, building data-pipelines, developing experimentation platforms, etc. Next, there is the lack of skills for experiment analysis. The common role in 2017 with statistical skills for experimentation is Data Scientist [18], [133]. In fact, skills for experimental design have been mentioned as one of the most critical demands for Data Scientists in a recent survey [18], and this position is expected to be the most difficult to fill in for software companies in the next 5-year period [134]. Third, there are organizational challenges such as agreeing on what to optimize for in experiments, or how to organize experimenters and analysts [9], [37].

Clearly, adopting or evolving experimentation remains challenging for many companies. Today, only a handful of companies succeeded in evolving their experimentation practices to the extent that allows them to run tens of thousands of experiments yearly (Amazon, Booking.com [135], Facebook [136], Google [58], LinkedIn [37], and Microsoft [11], to name a few).

In the majority of this thesis, specifically in Chapters 7-12, we aim to help the remainder of the software industry to advance their experimentation practices by sharing our distilled learnings based on the experience of companies that have succeed in this.

In the next section, we introduce our research approach.

3 RESEARCH APPROACH

In this section, we describe our research approach. We conducted a longitudinal case study research with several companies working in the embedded systems domain, and companies working within the online services and products domain. We start by first introducing the case study research methodology. Next, we introduce our case companies, and present the type of collaboration with them. Finally, we detail how we applied the described methodology in our research, and possible threats to validity to this work.

3.1 Case Study Research

In our research, we followed case study research guidelines presented by Runesson and Höst [137]. Case study is a research strategy used to investigate individual, group, organizational or social phenomena [138]. Typically this strategy is used to explain ‘how’ and answer ‘why’ type of questions, which require an extensive and in-depth understanding of the phenomenon being studied [138], [137]. Our research questions in this thesis aim for this type of understanding and individual studies attached to this thesis used linear-analytic approach [137] to case studies to construct new theories and knowledge about the phenomena under investigation.

Case study research can provide descriptions, test theories, or generate new theories. The case study strategy focuses on the dynamics within the setting and can involve either single or multiple cases, [137], [139]. Selection of cases is an important aspect of building theory from case studies. It is typical for case studies to

perform multiple techniques of data collection such as conducting interviews, writing field notes, conducting observations and focus groups, etc. [138], [139]. Both qualitative (e.g. interview statements) and quantitative (e.g. frequency of counts of some variable) types of data can be collected while conducting a case study. To understand and detail every case being studied, a case study typically involves within-case studies, which are central for insight generation and help researchers to cope with the often large quantities of data [139]. And by capturing the collection and analyses process over a period of time, a case study can be developed into a longitudinal case study [140].

3.2 Case Companies

In our research, we conducted a longitudinal multi-case study with companies working in the embedded systems domain and within the online services and products domain. In case study research, it is important to select cases that have suitable characteristics for answering the research questions under investigation. In most of the studies that we include in this thesis, we conducted multiple-case studies. The rationale was that by doing so, we would increase the reliability and validity of the results.

We engaged with the case companies in two different types of collaborations. First, and with the companies that primarily operate in the embedded systems domain, we collaborated within the Software Center collaboration. With companies that primarily work in the online domain we collaborated through internships and other types of engagements. We describe both types of engagement in greater detail next.

3.2.1 Software Center Companies Collaboration

Software Center is an European collaboration [141] of companies and universities that consists of a wide range of projects. At the moment of writing this thesis, there are five universities and eleven companies collaborating in Software Center on multiple projects at the same time. The universities are the following; Chalmers University, Gothenburg University, Malmö University, Linköping Univer-

sity and Mälardalen University. The companies, on the other hand, are the following: Axis Communications, Bosch, Ericsson, Grundfos, Jeppesen, Qamcom, Saab, Siemens, Tetra Pak, Volvo Cars, Volvo Trucks. As the name Software Center depicts, companies in this collaboration develop products and services that contain software. Each of the companies employs several thousands of practitioners and has multiple sites around the world. For the purpose of this research, we primarily collaborated with sites in the Nordics Region, and with two of the case companies in Germany.

Our longitudinal multiple-case study has been conducted as part of this established collaboration with ongoing work with one of the Software Center projects. In this project, we worked with seven case companies that are involved in large-scale development of software products. Although we met with other companies that are part of the collaboration as well (and are involved in the same project), we collaborated closely with seven of them. This is due to the availability of practitioners and the resources available. Four of the companies are based in Sweden and conduct work also in multiple sites around the world. The fifth company is based in Denmark, whereas the last two companies are based in Germany. We briefly describe the case companies next.

Company A is a provider of telecommunication systems and equipment, communications networks and multimedia solutions for mobile and fixed network operators. It is a multinational company with headquarters in Sweden. The company employs over hundred thousand employees worldwide and owns several thousands of patents. Throughout the longitudinal study, we met with representatives from their site in Sweden and in China.

Company B is a software company specializing in navigational information, operations management and optimization solutions. Their focus is in providing software solutions with accurate and timely navigational information. The company's headquarters are in the US and they have offices in more than fifty countries. Throughout the longitudinal study, we met with practitioners that primary work in Sweden.

Company C is a manufacturer and supplier of transport solutions, construction technology and vehicles for commercial use. They specialize in producing transport solutions that optimize their customers' needs in many different domains. The company's headquarters are in Sweden, however, their products are sold and used in most of the countries worldwide.

Company D is a world leading company in network video and offers products such as network cameras, video encoders, video management software and camera applications for professional IP video surveillance.

Company E is a manufacturer and supplier of pump systems. They produce circulator pumps for heating and air conditioning, as well as centrifugal pumps for transporting liquids. The company's headquarters are in Denmark. They employ about twenty thousand employees and for the purpose of this research, we met with representatives of the company from their headquarters office.

Company F develops software systems for several different domains, including healthcare, energy and infrastructure operations. Their portfolio is extensive and includes products and services for both B2B as well as B2C types of customers. They employ around 400.000 employees worldwide. We collaborated with one of their sites in Germany.

Company G is a multinational engineering and electronics company headquartered in Germany. They develop components for different types of vehicles, industrial machinery and tools for developing large systems. Similarly as company F, they also employ around 400.000 employees worldwide. We collaborated with one of their sites in Germany.

3.2.1.1 *Software Center Sprints*

The collaboration with the software center companies was conducted in 6 month sprints. In a typical sprint, we closely collaborated with a number of companies through workshops, meetings, interviews and other methods that we describe in greater detail in 3.3. During a typical sprint, company practitioners and university researchers with similar interests meet on a regular basis to identify current research challenges, advance on-going research, coordinate work, or to present and validate the findings. As research projects normally do not have immediate impact, maintaining organization interest is essential with both presenting on-going work as well as communicating results [142].

A typical sprint starts with a joint **“kick-off” workshop** session where case company practitioners and researchers meet to agree on a research focus of the sprint. Multiple companies present their challenges related to the project domain and an agreement on a common and most important topic is selected for the focus of the sprint. In addition, activities for the rest of the sprint are planned. The focus of the sprint is then further discussed on **individual workshops** with each of the case companies that wish to participate in the sprint. Researchers in the sprint typically follow-up the workshop with a number of **individual interviews** with the participants. Aggregated data is analyzed by the researchers and findings are validated with each of the case companies in a validating session. This session is typically near the end of the sprint and is in a form of a **reporting workshop** with all of the case companies. In addition to the research activities above, **Software Center days** are organized at least once per year where all of the companies and universities meet and discuss recent research and industry challenges.

3.2.2 Online Companies Collaboration

The case companies presented in the previous section provided the foundation for our research in the early stages of our research. By collaborating with them, we learned how they collect and use data to make better decisions, and also how they aim to improve to become faster and more accurate. Within this context, we identified a common denominator – experimentation. All of our case companies strive to build capacity to learn faster and more accurately through experimentation. To enable them to progress faster, and to provide scientific knowledge on how to adopt and grow experimentation, we expanded our research to companies that work outside of the embedded systems domain. Specifically, we established a collaboration with companies that primarily develop online products and services:

Case Company H is a large-scale multinational organization with headquarters based in the United States. The company develops software and hardware products, as well as offers many different types of services spanning from web services and mobile applications to system drivers. The company has over 100,000 employees. For the purpose of this research, we engaged with practitioners working at their main site in Washington state.

Case Company I is a large-scale online accommodation provider based in the Netherlands. The company has over 10,000 employees working across all of the continents. We engaged with practitioners working at their main site in the Netherlands.

Case Company J is a large-scale online travel-fare provider based in Scotland. The company has over 1,000 employees, and they work in several sites across Europe, Asia and US. We engaged with practitioners working at their main site in Scotland.

Case Company K: Our final case company is a large-scale business and financial software company based in the United States. The company has approximately 10,000 employees working at several sites across the US. For the purpose of this research, we engaged with practitioners working at the site located in California.

3.3 Data Collection

During our longitudinal study, we carried a number of activities to collect data. First, we conducted joint and individual workshops with our case companies. Second, we conducted open-ended interviews with individual practitioners. Third, we conducted reporting workshops and evaluation sessions. Finally, we conducted two internships at one of the case companies. We describe the format of each of the data collection activities next.

3.3.1 Joint Company Workshops

We conducted joint workshops with practitioners from several case companies. During the joint workshops we asked open-ended questions and took field notes by writing our observations and what respondents answered. The joint workshop served to build an understanding of the needs and challenges experienced by several companies, and to establish a common agreement between the companies and researchers on the research focus.

3.3.2 Individual Company Workshops

Individual workshops served as an activity where more company specific questions could be asked. Typically, individual company workshops followed the joint workshop. At every individual workshop, we conducted focus groups sessions, where topics of interest typically identified in the joint sessions or prior research were discussed. At each of the workshops, we took field notes that captured not only what the participants answered, but also the way they said it (e.g. the emotional responses behind their statements). During the individual workshops, practitioners frequently shared in-depth insights about the studied phenomena.

3.3.3 Reporting Workshops

We reported our results to all of the case companies that participated in our research through reporting workshops. In the context of Software Center collaboration, the reporting workshops occurred twice a year at the end of every sprint. With companies

working in the online domain we held individual reporting workshops (e.g. at the end of every internship, and per availability of practitioners). Reporting workshops were an opportunity to both communicate our analyzed results, but also to get feedback on our interpretation and the overall work that was conducted. In total, we conducted 7 reporting workshops.

3.3.4 Interviews

The interviews were an opportunity to collect data from practitioners from another angle. Due to the natural distribution of participants at workshops, some individuals might have abstained from speaking or sharing their thoughts. The interactive nature of the interviews was helpful to capture and develop interesting and unexpected themes brought-up by the interviewees in a private environment. Also, interviews can provide additional tacit information through gestures, voice tone, facial expressions etc., and are important for the interpretation of the expressed words.

The interviewers were chosen on the basis of their role, expertise and their relevance with respect to the investigation.

The typical interview format started with an introduction and a short explanation of the research being conducted. Participants were then asked on their experience with the focus of the research. Typically, we also asked for examples on successes, pitfalls, and pain points that they experience with the topic under investigation. We mostly used semi-structured interviews, in which we had pre-defined topics, and a few open-ended questions to lead the discussion. Following-up questions were frequently brought up us to probe the interviewees' statements with concrete facts. Semi-structured interviews were very useful for exploratory purposes [137] while for more confirmatory and evaluation purposes, structured interviews or questionnaires are more suitable.

During the interviews, the participants had the opportunity to describe processes and organizational features (useful for understanding the context), or simply share their perspective on the studied phenomenon on top of answering our questions.

3.3.5 Internships

The author of this thesis also conducted two internships at one of the case companies – Microsoft - in the greater Seattle area, USA. During the internships, we collaborated with the Analysis and Experimentation (A&E) team based in Bellevue. The A&E team consists of approximately 90 people, who provides a platform and a service for running controlled experiments for their customers – the product teams within Microsoft. Overall, they conduct over 10.000 experiments every year, engaging users of Bing, Xbox, Skype, Office and other products in different experiments every day. Throughout the internship time which lasted approximately 6 months, we collected historical data about experiments conducted in the last ten years, participated and took notes at daily meetings where recent experiment analyses were discussed, interviewed case company participants, collaborated with product teams, and conducted evaluation sessions of our work. The internships at Microsoft enabled us to conduct research work that goes beyond the traditional case study approach due to the magnitude of highly relevant knowledge that we could obtain, the skills of practitioners that we could learn from and that provided feedback on our work, and opportunities to realize our ideas and validate them in a practical setting.

3.3.6 Tutorials

In addition to the data collection techniques described above, the author of this thesis also conducted tutorials on experimentation. The tutorials were educational events focused on teaching the practitioners on how to build and evolve experimentation capabilities in their companies. A typical tutorial was 3 hours long, conducted in English, and was attended by between 10 to 25 practitioners. During the tutorial we shared our experience on evolving data-driven capabilities and collected additional data from the attending practitioners that described their context, challenges that they experience and solutions that work for them.

3.3.7 Prototyping

Some of our data that we used for the analysis in this thesis has been collected through prototyping sessions. For example, to help companies advance one of the dimensions of experimentation growth discussed in several of the papers included in this thesis, we developed and prototyped with an artefact, which we integrated to the companies' experimentation platform. We evaluated the prototype with a number of practitioners and improved it along the development process.

3.3.8 Questionnaires

Questionnaires are a tool that is especially applicable for confirmatory research, or to evaluate a certain phenomenon [137]. We used a questionnaire in two studies appended to this thesis. First, to evaluate how the company participants experienced the prototype that we implemented and integrated to one of the case companies systems. Second, in a study where we evaluated the current state-of-the-practice of experimentation.

Questionnaires suffer from construct validity threats. In both studies that use this data-collection technique, we ensured that the participants were familiar with the phenomena that was investigated. For example, the participants were solicited at conferences that are attended by practitioners that work in the field of the study. Our research was always presented to the participants upfront either through a tutorial or through research material.

3.3.9 Other data collection methods

In addition to the data collection methods described above, the author of this thesis participated at hackathons at two of the case companies, conducted participatory observations during numerous visits and stays at the case companies, and exchanged e-mails and skype messages.

We summarize the various data collection activities with respect to the results of our work through the included publications on Figure 2.

3.4 Data Analysis

We analyzed the collected data using a combination of qualitative and quantitative approaches, which contributed to apply methodological triangulation, as recommended in [143]. Qualitative analysis gives in-depth understanding of a process or a system through interviews, workshops etc., while quantitative analysis strengthens the results, and is used to test hypothesis. We describe both types of analysis next.

3.4.1 Qualitative Analysis

Most of the raw data that we collected through interviews, workshop notes, documentation or other techniques described in the previous section is of qualitative nature. To analyze it, we first transcribed it when necessary (for example, the interviews were recorded and transcribed after the data collection activity). Next, we read the qualitative data word by word in order to derive codes through an open coding approach. In this process, we first highlighted individual phrases that captured our attention in the transcriptions. We used color coding to highlight different topics that emerged during the analysis process. In this process, we reflected on the highlighted text several times and took notes. As this process continued for a few iterations (e.g. by analyzing the first few interviews), codes emerged from the highlights. In the next step, we sorted these codes into categories (e.g. grouping them by color). This type of design is appropriate when striving to describe a phenomenon where existing theory or research literature is limited. After we emerged with a few codes, we created definitions for those codes and continued to code the rest of the data using the definitions.

As we show on the example in Table 1 below, The high-level category “Use of feedback data” was further deducted into subcategories (“use purpose”, “use challenge”, and “use destination”). The advantage of this approach was that we gained information from study participants without imposing preconceived categories.

In the coding process, we frequently independently and then jointly analyzed the collected data by following the approach above, and derived the codes that were consolidated with an independent researcher that also participated in joint data-collection activities. Researchers also added their notes (*memos*) to codes which were used in the next iteration of the analysis process.

Raw Text	And internally use it as powerful information to improve our products and features and align our offer with the actual needs.
Code Tree	<div>USE OF FEEDBACK DATA</div> <div>USE PURPOSE</div> <div>USE CHALLENGE</div> <div>USE DESTINATION</div>

Table 1. Coding example.

Most of the analysis in this study was done with the intent to identify new topics through hypothesis generation or to detail and confirm our findings. We used *constant comparisons* [137] for this part (as indicated by the coding of the data first and an analytical approach to build new hypothesis next). The constant comparative method is designed to aid the analyst who possesses these abilities in generating a theory that is integrated, consistent, plausible, close to the data, and at the same time generalizable to a certain extent. When we extended our studies with other case companies and other practitioners we performed *hypothesis confirmations* [137]. In this way we could improve the theory and further validate our research contributions.

3.4.2 Quantitative Analysis

Some of the research conducted in this research also involved quantitative data analysis. Specifically, we computed descriptive and causal statistics.. For this purpose, we used Microsoft Excel and R with statistical packages as standard tools.

Most of the quantitative data analysis was conducted in order to aggregate the results or visualize them in some way. For example, and presenting survey results, we counted the frequency of times certain features are implemented. In one of the papers that are included in this thesis, we also conducted a controlled experiment where we used a standard t-test [129] to measure the level of statistically significant difference between our groups. Using the t-test, we computed a p-value. A p-value is the probability to observe a difference between the means of the two groups in the experiment equal to (or a larger one than the one actually observed) under the assumption that the two groups are not different. This was exactly the intent of the study that used this method, where we were evaluating whether there is a difference in the effectiveness between the group that used our artefact compared to the group that did not.

3.5 Threats to validity

Runeson and Höst [137] recommend the use of different kinds of triangulation in order to mitigate several kinds of threats to validity that exist in case study research. For example, using **confirming evidence from multiple sources** (*source triangulation* - getting the same answers from architects, engineers, and managers), using **different methodologies** (*methodology triangulation* - employing quantitative and qualitative methods), and **having several data collectors** (*observer-triangulation* - several researchers simultaneously participate in an event and take notes).

We applied all of the aforementioned triangulations in our research. For example, we used interviews, workshops and other techniques to collect data, we collected different types of data, and frequently multiple people were involved in the data collection and analysis process.

Despite our mitigations, however, our research risks on having several threats to validity. In the following subsections, we discuss how we mitigated the threats to validity that are applicable for this type of research [137]. These are the construct validity, internal validity, external validity, and reliability.

3.5.1 Construct Validity

Construct validity is the level of joint understanding of the phenomena being investigated, and it is of critical importance that the data collected is representative of the investigation. For example, when a research topic is being investigated through interviews, it is critical that both parties align on the topic of investigation. We mitigated this threat in a number of steps.

First, we started and conducted this research with participants that were either already a part of the **ongoing collaboration**, or they became involved through our workshops and other collaborations. This enabled us to ensure a **prolong involvement** [143] with practitioners. In this way, we could establish trustworthy environment in which **concepts** from academia were spread and **understood** by the industrial representatives. At the same time, industrial jargon that differs between organizations, was better understood by the researchers.

Second, we always conducted data collection with practitioners that **work in different** roles and that have **various levels of experience**. This enabled us to study the problem from viewpoints that differ in several ways. Any misinterpretation that occurred at individual data collection sessions could be efficiently verified by triangulating it with the other sessions, or by asking the company contacts for clarifications.

Third, and at beginning of every data collection activity, we **introduced the concepts under investigation** to the participants along with the terminology that is being used. This could be considered conservative, however, aligning on the topic being investigated provided many benefits along the data collection sessions (for example, it provided an ability to better focus the sessions).

3.5.2 Internal Validity

Threats to internal validity are present when investigating a direct cause-effect relationship. In one of the studies attached to this thesis we studied a direct cause-effect of introducing an artefact into experiment analysis. Specifically, we were interested if experiment analysis that used our artefact together with the existing system performed a more comprehensive and effective analysis of experiments, compared to the experiment analysis instances that used only the existing system. We mitigated the threats to internal validity of this study in several steps.

First, we designed the study as a controlled experiment where two options of experiment analysis were possible (with artefact and without). We randomly alternated the two options for every participant, giving every participant the ability to conduct an experiment analysis through both approaches multiple times. The experiments were given to participants with some context, and in order that prevented learning effects. For example, consecutive experiments differed on, at least, the product type, available segmentation, or types of metric movements that occurred.

Next, the participants were not aware of the group that they were in, neither of the exact measures that we collected. The criterion for every metric increase was defined upfront by the first two authors of the study, and the study was piloted on one Data Scientist. Every data point contributing to a metric (e.g. every time a study participant mentioned a certain insight outload) was logged with its time of appearance both during the session itself and triangulated with the recording after the session.

Finally, we triangulated the quantitative results obtained by computing a p-value through a standard t-test with a questionnaire, to confirm the causal effect of our artefact.

3.5.3 External Validity

This aspect of validity is concerned with the extent to which it is possible to generalize the findings, and to what extent the findings are of interest to other people outside the investigated case [137]. In case study research, we typically do not generalize to the overall population, but instead try to analytically reason to what extent are the findings applicable to other cases that have similar characteristics as the ones that were part of the study. To improve the external validity of our research, we conducted a number of steps in our research.

First, theory building in our case studies followed an inductive analysis of data obtained from practitioners working in different roles and with varying levels of expertise. Next, we collaborated with companies that are at various levels of maturity in respect to the phenomena that we studied, frequently extending our initial theory with confirmation studies with more cases.

Next, the case companies that we collaborated with work in many different domains, including companies that are in the forefront of automotive, network equipment, online search, productivity tools, etc. The ability to discuss the same phenomena in many different domains improves the external validity.

3.5.4 Reliability

Reliability addresses researcher bias. It is important to address this threat in order to understand if the data and analysis are dependent on a specific researcher or if the same study would lead to the same results by other researchers. This threat to validity is especially present when qualitative data is collected, since there are several points in which the researcher's bias might be introduced in the results.

To mitigate reliability threats, there were always multiple researchers reviewing the results of a study, as well as multiple practitioners from our case companies evaluating our findings. This member checking technique [143] was executed during interviews and workshops, and was one of the main focus of the preliminary findings reporting workshops.

3.6 Ethical Remarks

The knowledge presented in this thesis enables companies to build platforms and tools to effectively learn about their customers and users. This can have ethical implications.

The author of this thesis believes that as technology experts, we have the responsibility to not only improve the tools and build new platforms for data collection and analysis, but also to consider the aspects of their use for good. Consider for example the recent emotional experiments at Facebook [144] briefly discussed in 8, the incidents with Cambridge Analytica with respect to the 2016 US presidential election interference, or any other related data incident. The data is quickly becoming the epicenter of many business models, and suitable precautions need to be considered in how it is being handled [145].

There are a number of ethical frameworks [146] that can be used to argue against or towards the use of data for the purpose of software product improvement. We take the consequence-based (utilitarian) viewpoints to argue for investing into the work described in this thesis.

In consequence-based (utilitarian) view [146], the morally correct action is the one that produces the most desired outcome. In our view and experience, it is as beneficial for the users of software products as it is for the companies to receive and develop experience that maximizes user needs. Moreover, and as we stress the importance of experimentation, decisions are made on aggregated samples in favor of the greatest number of individuals (for example, when two ranking algorithms are evaluated, the one that produces valuable results for a greater number of users will be deployed). The social utility (for example, user happiness) is therefore maximized through the use of data. Despite the positive implications, users of software product should, however, be informed about the data that is being collected about their activity when they use the product, and have the option to control and delete the data if they desire so.

4 COLLECTING FEEDBACK DATA

Abstract

With agile teams becoming increasingly multi-disciplinary and including all functions, the role of customer feedback is gaining momentum. Today, companies collect feedback directly from customers, as well as indirectly from their products. As a result, companies face a situation in which the amount of data from which they can learn about their customers is larger than ever before. In previous studies, the collection of data is often identified as challenging. However, and as illustrated in our research, the challenge is not the collection of data but rather how to share this data among people in order to make effective use of it. In this chapter, and based on case study research in three large software-intensive companies, we (1) provide empirical evidence that ‘lack of sharing’ is the primary reason for insufficient use of customer and product data, and (2) develop a model in which we identify what data is collected, by whom data is collected and in what development phases it is used. In particular, the model depicts critical hand-overs where certain types of data get lost, as well as the implications associated with this. We conclude that companies benefit from a very limited part of the data they collect, and that lack of sharing of data drives inaccurate assumptions of what constitutes customer value.

This Chapter is based on

A. Fabijan, H. H. Olsson, and J. Bosch, "The Lack of Sharing of Customer Data in Large Software Organizations: Challenges and Implications," in Proceedings of the 17th International Conference on Agile Software Development XP'16, (Edinburgh, Scotland, April 2016), 39–52.

4.1 Introduction

Traditional ‘waterfall-like’ methods of software development are progressively being replaced by development approaches such as e.g. agile practices that support rapid and continuous delivery of customer value [53]. Although the collection of customer feedback has always been important for R&D teams in order to better understand what customers want, it is today, when R&D teams are becoming increasingly multi-disciplinary to include all functions, that the full potential of customer data can be utilized [52]. In recent years, increasing attention has been put on the many different techniques that companies use to collect customer feedback. With connected products, and trends such as ‘Big Data’ [20] and ‘Internet of Things’ [147], the qualitative techniques such as e.g. customer surveys, interviews and observations, are being complemented with quantitative logging and automated data collection techniques. For most companies, the increasing opportunities to collect data has resulted in rapidly growing amounts of data revealing contextual information about customer experiences and tasks, and technical information revealing system performance and operation.

However, and as recognized in recent research [42], the challenge is no longer about how to collect data. Rather, the challenge is about how to make efficient use of the large volumes of data that are continuously collected and that have the potential to reveal customer behaviors as well as product performance [20], [148], [149]. Although having access to large amounts of data, most companies experience insufficient use of the data they collect, and as a result weak impact on decision-making and processes.

In this chapter, we explore data collection practices in three large software-intensive companies, and we identify that ‘lack of sharing’ of data is the primary reason for insufficient use and impact of collected data. While the case companies collect large amounts of data from customers and from products in the field, they suffer from lack of practices that help them share data between people and development phases. As a result, decision-making and prioritization processes do not improve based on an accumulated data set that

evolves throughout the development cycle, and organizations risk repetition of work due to lack of traceability.

The contribution of the chapter is twofold. First, we identify that ‘lack of sharing’ is the primary reason for insufficient use of data and we provide empirical evidence on the challenges and implications involved in sharing of data in large software organizations. Second, and based on our empirical findings, we develop a model in which we identify what data is collected, by whom data is collected and in what development phases it is used. Our model depicts critical hand-overs where certain types of data get lost, and how this causes a situation where data does not accumulate and evolve throughout the development process. By capturing ‘current state-of-practice’, and by identifying critical hand-overs where data gets lost, the model supports companies in identifying what challenges they experience, and what implications this will result in. The awareness that the model helps create can work as valuable input when deciding what actions to take to improve sharing of data in large software-intensive organizations.

4.2 Background

4.2.1 Collection of Customer Feedback

In most companies, customer feedback is collected on a frequent basis in order to learn about how customers use products, what features they appreciate and what functionality they would like to see in new products [73], [148]. Typically, a wide range of different techniques are used to collect this feedback, spanning from qualitative techniques capturing customer experiences and behaviors [59], [149] to quantitative techniques capturing product performance and operation [150], [151]. While the qualitative techniques are used primarily in the early stages of development in order to understand the context in which the customer operates, the quantitative techniques are used post-deployment in order to understand the actual usage of products.

Starting with the pre-development stage, companies typically collect qualitative customer feedback using customer journeys, interviews, questionnaires and surveys [59], [149], forming the basis for

the requirements generation [13]. At this stage, contextual information on the purpose of the product or a feature with functional characteristics and means of use are typically collected by customer representatives. Typically, this information is used to both define functional requirements as well as to form customer groups with similar needs and priorities, also known as personas.

During development, customer feedback is typically collected in prototyping sessions in which customers test the prototype, discuss it with the developers and user experience (UX) specialists, and suggest modifications of e.g. the user interface [59], [149], [152]. As a result, developers get feedback on product behaviors and initial performance data. Customer feedback is typically mixed and consists of both qualitative information on e.g. design decisions and quantitative operational data.

In the post-deployment stage, and when the product has been released to its customers, a number of techniques are used to collect customer and product data. First, and since the products are increasingly being connected to the Internet and equipped with data collection mechanisms, operational data, and data revealing feature usage is collected [59], [152], [153]. Typically, this data is of quantitative type and collected by the engineers that operate the product and service centers that support it. Second, if customers generate incident requests and attach the product log revealing the state of the product, error message and other details. These are important sources of information for the support engineers when troubleshooting and improving the product [40]. Also, and as recognized in previous research [154], A/B testing is a commonly deployed technique to collect quantitative feedback in connected products on which version of the feature offers a better conversion or return of investment. And although increasing amounts of data are being collected, very little is actually being used. The challenges in aggregating and analyzing this data in an efficient way prevent higher levels of the organization from benefiting from it [150].

4.2.2 Impact and Use of Customer Data

Companies operating within transportation, telecommunications, retailing, hospitality, travel, or health care industries already today gather and store large amounts of valuable customer data [147]. These data, in combination with a holistic understanding of the resources needed in customer value-creating processes and practices, can provide the companies that fully utilize it a competitive advantage on the market [155].

However, challenges with meaningfully combining and analyzing these customer data in an efficient way are preventing companies from utilizing the full potential from it [20], [42]. Instead of a complete organization benefiting from an accumulated knowledge, it is mostly only the engineers and technicians that have an advantage in using this data for operational purposes [150]. Higher levels in the organization such as product management or customer relationship departments need to find ways of better utilizing customer data in order to unlock its potential and use it for prioritization and customer value-actualization processes [155].

4.3 Research Method

This research builds on an ongoing work with three case companies that use agile methods and are involved in large-scale development of software products. The study was conducted between August 2015 and December 2015. We selected the case study methodology as an empirical method because it aims at investigating contemporary phenomena in their context, and it is well suited for this kind of software engineering research [137]. This is due to the fact that objects of this study are hard to study in isolation and case studies are by definition conducted in real world settings.

Based on experience from previous projects on how to advance beyond agile practices [45], [46], we held three individual workshops with all the companies involved in this research, following up with twenty-two individual interviews. We list the participants and their roles in Table 2.

Company	Representatives	
Company A	1 Product Owner	1 Release Manager
	1 Product Manager	1 Area Prod.
	2 System Managers	Mng.*
	2 Software Engineer	1 Lean Coach*
		1 Section Mng.*
Company B	1 Product Owner	
	1 System Architect	
	1 UX Designer	
	1 Service Manager	
Company C	1 Product Owner	1 Feature Coord.
	2 Product Strategists	1 Sales Manager
	2 UX Managers	2Technology Spec.
	2 Function Owners	

Table 2. Participating companies and their representatives.

4.3.1 Data Collection

During the group workshops with the companies, we were always three researchers sharing the responsibility of asking questions and facilitating the group discussion. Notes were taken by two of the researches and after each workshop, these notes were consolidated and shared to the third researcher and company representatives.

First, we conducted a workshop with an exercise with the post-it notes that build our inventory of the customer feedback techniques. Second, we held semi-structured group interviews with open-ended questions [2] during the workshop. These questions were asked by on of the researcher while two of the researchers were taking notes. In addition to the workshops, we conducted twenty-two individual interviews that lasted one hour in average, and were recorded using an iPhone Memo application. Individual Interviews were conducted and transcribed by one of the researchers. In total, we collected 13 pages of workshop notes, 176 post-it notes, 138 pages of interview transcriptions, and 9 graphical illustrations from the interviewees. All workshops and individual interviews were conducted in English.

4.3.2 Data Analysis

During analysis, the workshop notes, post-it notes, interview transcriptions and graphical illustrations were used when coding the data. The data collected were analyzed following the conventional qualitative content analysis approach where we derived the codes directly from the text data. This type of design is appropriate when striving to describe a phenomenon where existing theory or research literature is limited. Two of the researchers first independently and then jointly analyzed the collected data and derived the final codes that were consolidated with the third and independent researcher who also participated at the workshops. As soon as any questions or potential misunderstandings occurred, we verified the information with the other researcher and participating representatives from the companies.

4.3.3 Validity considerations

To improve the study's construct validity, we conducted the exercise with the post-it notes and semi-structured interviews at the workshops with representatives working in several different roles and companies. This enabled us to ask clarifying questions, prevent misinterpretations, and study the phenomena from different angles. Next, we combined the workshop interviews with individual interviews. Workshop and interview notes were independently assessed by two researchers, guaranteeing inter-rater reliability. And since this study builds on ongoing work, the overall expectations between the researchers and companies were aligned and well understood.

The results of the validation cannot directly translate to other companies. However, considering external validity, and since these companies represent the current state of large-scale software development of embedded systems industry [3], we believe that the results can be generalized to other large-scale software development companies.

4.4 Findings

In this section, we present our empirical findings. In accordance with our research interests, we first outline the generalized data collection practices in the three case companies, i.e. what types of data that is collected in the different development phases, and by whom. Second, we identify the challenges that are associated with sharing of data in these organizations. Finally, we explore their implications.

4.4.1 Data Collection Practices: Current State

In the case companies, data is collected throughout the development cycle and by different roles in the organization. Typically, people working in the early phases of development collect qualitative data from customers reflecting customer environments, customer experience and customer tasks. The later in the development cycle, the more quantitative data is collected, reflecting system performance and operation when in the field. In Tables Table 3 - Table 5, we illustrate the data collection practices, together with the customer feedback methods and types of data that different roles collect in each of the development stages.

	Roles that collect customer feedback	Common customer feedback collection techniques	Common types of customer feedback collected
Pre-Development	Strategy specialists, Product managers, Product owners	Reading of industry press, Reading of published standards, Reading of internal reports, Reading customer visit reviews	Customer wishes, Short/Long market trends, Competitors ability of delivering the product
	Strategy specialists, Feature owners	Telephone interviews, Face-to-face interviews, Conducting group interviews	Existing product satisfaction, Future product specification, Personas and User Journeys

Table 3. Customer data collection practices in the pre-development stage.

	Roles that collect customer feedback	Common customer feedback collection techniques	Common types of customer feedback collected
Development	UX specialists, Software Engineers	System Usability Scale Form, Asking open ended questions, Demonstrating prototypes, Filming of users' product use	Acceptance of the prototype, Eye behavior and focus time, Points of pain, Bottlenecks and constraints, Interaction design sketches
	System managers, System architects, Software engineers	Consolidate feedback from other projects, Reading prototype log entries	Small improvement wishes, Configuration data, Product operational data

Table 4. Customer data collection practices in the development stage.

	Roles that collect customer feedback	Common customer feedback collection techniques	Common types of customer feedback collected
Post-Deployment	Release managers, Service managers Software engineers	Reading of customer reports, Analyzing incidents, Aggregating customer requests, Analyzing product log files	Number of incid. and req., Duration of incid. and req., Product operational data, Product performance data
	Sales managers	Reading articles in the media, Sentimental analysis Customer events participation, Reading industry press, Performing trend analysis	Opinions about the appeal of the product, Performance of the product, Business case descriptions

Table 5. Customer data collection practices in the post-deployment stage.

4.4.2 Data Sharing Practices: Challenges

Based on our interviews, we see that there are a number of challenges associated with sharing of data in large organizations. For example, our interviewees all report of difficulties in getting access to data that was collected by someone else and in a different development phase. Below, we identify the main challenges associated with sharing of data in the case companies:

- ***Fragmented collection and storage of data***

Individuals independently collect increasing amounts of customer feedback, analyze the data they obtained, and store their findings on local repositories. Although these findings are occasionally presented at meetings, the lack of transparency and tools prevents

others in the organization to use and benefit from the data. With so many different roles collecting and storing data, systematic sharing across development phases becomes almost impossible. As a result, only those roles that work in close collaboration share data, and benefit from the analysis of this data. This situation is illustrated in the following quotes:

“... it is all in my head more or less.” -Product owner, Company B
“Information exists but we don’t know where it is.”

–UX Specialist from Company C

“I do not know everyone... So I contact only the person who is next in line.”
–Sales manager from Company C.

- ***Filtering of customer data***

People collect data, and share it only within the development stage they typically work in. For example, practitioners in the development phase actively exchange product log data, interaction design sketches, quality statistics and trouble reports. Similarly, those working in the post-deployment phase exchange release notes, business case descriptions and management system issues. Attempts to communicate the significance of customer feedback and their findings across development stages are typically unsuccessful. Feedback that is shared is filtered quantitative data.

“It is like there is a wall in-between. There is a tradition that we should not talk to each other.” -Product Owner from Company C.

- ***Arduous to measure means hard to share.***

The only data that is successfully shared among people and development phases, is quantitative data representing those things that can be easily measured such as e.g. system performance and operation. The case companies are successful in sharing transaction records, incident figures, feature usage data and other technical feedback that can be easily measured. However, qualitative data such as user stories, feature purpose, or the intention of a certain requirement typically stay with the people that collected that feedback. As a result, and instead of benefitting from an accumulated set of data that evolves over time, companies run the risk of using fragmented data sets that misrepresent the customer and provides an insufficient understanding of what constitutes customer value.

“Maybe 10% of information is shared. It is very difficult. It takes so much time, to, you need to write a novel more or less and distribute it”

-Product manager from Company A.

4.4.3 Data Sharing Practices: Implications

Due to very limited amounts of data being shared among people and across the development phases, the case companies experience a number of implications. Below, we present the implications:

- ***Non-evolving and non-accumulating data.***

Although quantitative data describing operational and performance requirements is typically shared, the lack of qualitative information with the context describing where, how and why a certain feature or a product is needed and how it will be used cause discrepancies in understanding the overall purpose. As a result, the data forming customer knowledge across the development stages does not accumulate and evolve. Consequently, practitioners developing the product do not fully understand the overall purpose of the product or a feature under development and develop suboptimal products that can be different from the customer wishes.

*“I think now a lot of thing are developed in a sub optimized way.”
-Technology Spec. from company C.*

*“We get feature which is broken down and then this value somehow got lost when it was broken down, then it is harder to understand what they really need it for.”
-Software engineer from Company B.*

- ***Repetition of work.***

Due to the lack of access to the qualitative feedback that is collected in the early stages of development, roles in later stages that seek contextual understanding of a feature are sometimes required to collect identical feedback to the one that was already collected. Consequently, resources are spent on repeating the work that has already been done once.

“You cannot build on what is already there since you don’t know. You then repeat an activity that was already made by someone else.” -UX specialist from Company C.

- ***Inaccurate models of customer value.***

Since the qualitative customer feedback is not shared across the development phases, companies risk to use only the available quantitative customer feedback to build or update the understanding of the customer. This results in inaccurate assumptions on what constitutes customer value. And as a consequence of using the feedback for prioritization on the product management level, projects that create waste risk to get prioritized.

“You think one thing is important but you don’t realize that there is another thing that was even more important.” -Technology Spec. from company C.

- ***Validation of customer value is a “self-fulfilling prophecy”.***

Due to the fact that only quantitative customer feedback is exchanged across the development phases and development organization, companies risk to validate their products using only the effortlessly quantifiable feedback, and neglecting the rest. Instead of using the accumulated customer feedback and holistically assess their products, the validation of customer value becomes a “self-fulfilling prophecy” in that it focuses on developing and verifying things that can be quantified and provide tangible evidence.

Challenge	Description	Company implications	Product implications
Fragmented collection and storage of data	Sharing of data is limited across the development stages.	No evolving and accumulating of customer data and understanding.	Suboptimal products are being developed.
Filtering of customer data.	Only roles that work in close cooperation exchange feedback.	Inaccurate assumptions on customer value and repeating work.	Risk of developing wasteful features.
Arduous to measure means hard to share.	What can easily be measured and quantified is shared.	Validation of customer value is a “self-fulfilling prophecy”.	Product maximizes partial models of customer value.

Table 6. The mapping of identified challenges to their implications.

4.5 Discussion

Multi-disciplinary teams involved in the development of a software product are using customer feedback to develop and improve the product or a feature they are responsible for. Previous research [42], [59] and our empirical findings show that companies collect increasingly large amounts of customer data. Both using the qualitative techniques are used primarily in the early stages of development [42], [59], [150] to construct an understanding of the customer and the context

they operate in, and quantitative techniques that are used post-deployment to monitor the actual usage of products in the field [40], [150]. And although companies gather and store large amounts of valuable customer data [147], challenges with meaningfully combining and analyzing it in an efficient way [20], [42] are preventing companies from evolving the data across the development stages and accumulating the customer knowledge.

4.5.1 The Model: From Quantifiable Feedback to Partial Customer Value

In response to the challenges and implications presented above, we illustrate our findings and challenges in a descriptive model on Figure 1.

In the development process, the model advocates an approach in which an internal model of customer value in companies is being created. We illustrate that companies in fragments collect a complete understating of the customer and their wishes, however, benefit only from a part of the understanding.

In our model, we distinguish between three development stages, i.e. pre-development, development and post-deployment. Although we recognize that this is a simplified view, and that most development processes are of an iterative nature, we use these stages as they typically involve similar roles, techniques, and types of feedback collected.

On Figure 3, we list a few roles that collect customer feedback (A) and different methods of how they perform the collection (B). Within each of the development stages we list the types of feedback being shared across the stages with a solid green lines and those that are not with a dashed red lines. Between development stages we identify three critical hand-over points where customer data that could and should get shared, dissipates. Instead of accumulating data being handed over, gaps across stages appear (illustrated with “?”symbols in blocks on Figure 3).

Figure 3. Customer feedback sharing practices model.

4.5.1.1 *The Vaporization of Customer Data.*

We identify three critical hand-over blocks that cause data to disappear and prevent practitioners on project to build-on.

(1) *The PreDev Block:* While there is extensive collection of qualitative customer feedback such as user journeys and product satisfaction surveys (Illustrated with C on Figure 1), roles working in the pre-development stage do not sufficiently supply the development part of the organization with the information they collect. Qualitative data that would inform the development stage on the context of the product under development, how it is going to be used, and who the different user groups perishes in the hand-over process between product owners and managers on one side, and software engineers and UX specialist on the other (Illustrated with D on Figure 1). Specifically, personas, user journeys and customer wishes are the types of feedback that should be handed over to the development stage, however, they are not. Consequently, the development part of the organization is forced to repeat collection activities in order to obtain this information when in need, or continue developing the product following only the specifications / requirements that were handed to them.

(2) *The DevOps Block:* UX specialists and software engineers collect feedback on prototypes and their acceptance, as well as

where the constraints are. However, this information is only used within the development stage. As a consequence of not handing it over to the post-deployment stage service managers and software engineers (Illustrated with E on Figure 1), operators of the product do not understand the reason behind a certain configuration when solving a problem, and at the same time, suggest alternative solutions that were already known to be unacceptable to the developers.

(3)*The OpsDev Block*: In the post-deployment stage, release and service managers collect and exchange operational and performance data, however, do not share it with the development stage to software engineers and system managers. (Illustrated with F on Figure 1). This prevents the roles in the development stage such as system architects from e.g. deciding on an optimal architecture for a certain type of product and customer size.

4.5.1.2 *Unidirectional Flow of Feedback.*

Illustrated with red and dashed arrows on Figure 1, the flow of feedback from the earlier stages of the development to the ones in the later stages is very limited. On the other hand, the flow of feedback from the later stages to the early ones is extensive. This both supports our finding about extensive sharing of quantitative data, which is typically available in the later stages, as well as implies that it is easier to share data about earlier releases of the software under development compared to sharing feedback about the current release. Validating the value of the current release is consequently done very late.

4.5.1.3 *Shadow Representation of Customer Value.*

In the absence of the accumulated data being accessible and shared across the development stages (illustrated with missing data symbol “?” on Figure 1), people in later stages base their prioritizations and decisions on shadow beliefs existing in the organization. Consequently, and instead of having a unique understanding of what constitutes customer value, individual development stages and roles

prioritize based on their best intuition and shared quantitative data. If sharing of customer data in the direction towards the later stages is enabled, roles across the development stages will be able to conduct data-informative decisions. As seen in our findings, hazards of being purely quantitative data-driven are extensive. And with qualitative data being as accessible as quantitative, validation of customer data could be coherent, not a ‘self-fulfilling prophecy’ as it is today.

4.6 Conclusion

By moving away from traditional waterfall development practices and with agile teams becoming increasingly multi-disciplinary and including all functions from R&D to product management and sales [52], the role of customer feedback is increasingly gaining momentum. And although the collection of data has previously been identified as challenging, we show in our research that the challenge is not its collection, but rather how to share this data in order to make effective use of it.

In this chapter, we explored the data collection practices in three large software-intensive organizations, and we identify that lack of sharing of data is the main inhibitor for effective product development and improvement. Based on our case study findings, we see that currently (1) companies benefit from a very limited part of the data they collect due to a lack of sharing of data across development phases and organizational units, (2) companies form inaccurate assumptions on what constitutes customer value and waste resources on repeating the activities that have already been performed, and (3) validation of customer in companies today is a “self-fulfilling prophecy” in that it focuses on quantifiable things that provide tangible evidence.

5 DIFFERENTIATING FEATURE REALIZATION

Abstract

Software is no longer only supporting mechanical and electrical products. Today, it is becoming the main competitive advantage and an enabler of innovation. Not all software, however, has an equal impact on customers. Companies still struggle to differentiate between the features that are regularly used, there to be for sale, differentiating and that add value to customers, or which are regarded commodity. Goal: The aim of this chapter is to (1) identify the different types of software features that we can find in software products today, and (2) recommend how to prioritize the development activities for each of them. Method: In this chapter, we conduct a case study with five large-scale software intensive companies. Results: Our main result is a model in which we differentiate between four fundamentally different types of features (e.g. 'Checkbox', 'Flow', 'Duty' and 'Wow'). Conclusions: Our model helps companies in (1) differentiating between the feature types, and (2) selecting an optimal methodology for their development (e.g. 'Output-Driven' vs. 'Outcome-Driven').

This Chapter is based on

*A. Fabijan, H. H. Olsson, and J. Bosch, "Differentiating feature realization in software product development," in **Proceedings** of the 18th International Conference on Product-Focused Software Process Improvement, PROFES'17, (Innsbruck, Austria, December 2017), 221–236.*

5.1 Introduction

Rapid delivery of value to customers is one of the core priorities of software companies [4]–[6]. Consequently, the amount of software added to products with attempts to deliver the value is rapidly increasing. At first, software functionality was predominately required in products to support tangible electrical, hardware and mechanical solutions without delivering any other perceptible value for the customers. Organizations developed the software as a necessary cost, leaving the prioritization part to the responsible engineering, hardware and mechanical departments, without exploring the value of software features as such. Today, products that were once built purely from hardware components such as e.g. cars and household appliances, contain functionality that allows them to connect to the Internet, exchange information and self-improve over time. Software functionality is rapidly becoming the main competitive advantage of the product, and what delivers value to the customers [2].

However, the way in which software features are being developed, and how they are prioritized is still a challenge for many organizations. Often, and due to immaturity and lack of experience in software development, companies that transitioned in this way (e.g. from electrical to software companies) treat software features similarly to electronics or mechanics components. They risk being unable to identify what features are differentiating and that add value to customers, and what features are regarded commodity by customers. As a consequence of this, individual departments continue to prioritize what they find the most important and miss the opportunities to minimize and share the investments into commodity features [104], [46]. Companies that are becoming data-driven, are exploring this problem and trying to learn from the feedback that they collect to optimize the prioritization decisions. The focus of software companies and product team that recognized the benefits of being data-driven is to develop the product that delivers the most value for the customer [4]–[6]. For every feature being developed, the goal is to create clarity in what to develop (Value Identi-

fication), develop it to a correct extent (Value Realization), and accurately measure the progress (Value Validation).

In our work, we perform an exploratory case study in which we were interested in (1) identifying how companies differentiate between different types of features and (2) how they prioritize the development activities with respect to the type of the feature that they are developing. In this study, we identify that the lack of distinguishing between different types of features is the primary reason for inefficient resource allocation that, in the end, make innovation initiatives suffer. In our previous work [156], we confirmed that differentiating between the different levels of functionality is indeed a challenge that software companies face today. In this chapter, however, we further detail the feature differentiation model and evaluate it in two additional case companies.

The contribution of this chapter is twofold. First, we provide detailed guidelines on how to distinguish between different types of features that are being developed and we provide empirical evidence on the challenges and implications involved in this. Second, we detail a conceptual model to guide practitioners in prioritizing the development activities for each of the feature types identified in our differentiation model. We identify in our work that for the successful development of innovative ‘Wow’ and ‘Flow’ features, a different methodology is required. We label this methodology ‘Outcome-Driven development’ and demonstrate how it differs from ‘Output-Driven approach. With our contribution, companies can develop only the amount of feature that is required for commoditized functionality and, on the other hand, frees the resources to maximize their investments in innovative features that will deliver the most value.

5.2 Background

Software companies strive to become more effective in delivering value to their customers. Typically, they inherit the Agile principles on an individual development team level [12] and expand these practices across the product and other parts of the organization [52]. Next, they focus on various lean concepts such as eliminating

waste [95], [96], removing constraints in the development pipeline [94], and advancing towards continuous integration and continuous deployment of software functionality [53]. Continuous deployment is characterized by a bidirectional channel that enables companies not only to deliver new updates to their customers in order to rapidly prototype with them [31], but also to collect feedback on how these products are used. By evaluating ideas with customers, companies learn about their preferences. In this process, the actual product instrumentation data (for example, features used in a session) has the potential to identify improvements and make the prioritization process of valuable features more accurate [157]. In fact, a wide range of different techniques is used to collect feedback, spanning from qualitative techniques capturing customer experiences and behaviors [40], [75], [148], to quantitative techniques capturing product performance and operation [10], [57].

However, this development of getting rapid feedback in real time is only possible today when software products are increasingly becoming connected to the internet. And as such data focuses on what customers do rather than what they say, it complements other types of feedback data [90] and improves the understanding of the value that the product provides [45]. For example, knowing which features are used in a certain context helps companies in identifying customer preferences and possible bottlenecks.

The intuition of software development companies on customer preferences, can be as much as 90% of the time inaccurate [32], [47], [48]. In most companies, customer feedback is collected on a frequent basis in order to learn about how customers use products, what features they appreciate and what functionality they would like to see in new products [148]. This enables the companies to very quickly test new variants with the customers and explore how they perform with the respect to the metrics that are important. This concept is known as Continuous Experimentation [9] and its impact is extensive [25].

5.2.1 Business and Design Experimentation

Davenport [55] suggests the design of smart business experiments to emphasize the need to bridge business needs closer to software design and engineering. The importance of running experiments to learn more about customer behavior is one of the core principles of the Lean Startup methodology [31], [158]. This idea is emphasized also in the product management literature by Bosch [40] who propose the need of constant generation of new ideas that should be evaluated with customers. Fagerholm et al. [4] suggest the RIGHT model for continuous experimentation and support constant testing of ideas. In their view, this is essential to create the evaluation of product value as an integral part of the development process. The strength of experimentation is further accelerated with controlled experiments. In controlled experiments (also known as A/B test), users of a software product or a feature are randomly divided between the variants (e.g., the two different designs of a product interface) in a persistent manner (a user receives the same experience at multiple software uses). Users' interactions with the product are instrumented and key metrics are computed [29], [56]. Research contributions with practical guides on how to create controlled experiments in software product development have previously been published both by Microsoft [154], [57] and Google [58]. The Return on Investment (ROI) of controlled experiments has been demonstrated and discussed a number of times in the literature [56], [154].

5.2.2 Feature Differentiation

Already in 1984, Kano [86] has identified the need to differentiate between product attributes (e.g. attractiveness, satisfaction, must-be, indifferent and reverse attributes) based on their importance for the product and the stakeholders. One of the key characteristics of the Kano model itself is the notion of attribute drift (for example, an attribute such as camera resolution is expected to improve over time). Attribute drift is driven by customers' expectations and changes over time. In this context, Bosch [159] developed the 'Three Layer Product Model' (3LPM) on the idea that a return on

innovation requires differentiation [160]. The model provides a high-level understanding of the three different layers of features, i.e. commodity, differentiating and innovative, and it has been constructed for the purpose of reducing architectural complexity. And although these models can be applied to an industrial context, they do not provide guidance on how to differentiate between the different types of features, neither how to prioritize software development activities based on the differentiation. At the same time, the pace at which we are becoming to be able to learn from the customers in the software industry is faster than anywhere else [57], [148]. The feedback that is available today (e.g. feature usage data while the product is in the hands of the users) opens new possibilities for understanding and differentiating features in products. Using existing models (e.g. such as the Kano model or the 3LPM model) and accommodating new dimensions of learnings that we discussed above is challenging for software companies, and it blocks them from fully benefiting from the new possibilities [10], [156]. The implications of limited customer learning are substantive and can cause companies to inaccurately prioritize feature development by e.g. develop the software features to the wrong extent in either direction.

In our previous research [156], we developed a model in which four fundamentally different types of features are being developed. We name them “duty”, “wow”, “checkbox” and “flow” types of features. With “duty”, we label the type of features that are needed in the products due to a policy or regulatory requirement. “Checkbox” features are the features that companies need to provide to be on par with the competition that provides similar functionality. With “wow”, we label the differentiating features that are the deciding factor for buying a product. Finally, and with “flow”, we label the features in the product that are regularly used.

In this chapter, we address the new possibilities of learning from customers to (1) detail the differentiation of features for software products, and (2) to suggest on how to prioritize the development activities for each of them. Our model (1) helps companies in dif-

ferentiating between the feature types, and (2) selecting a methodology for their development (e.g. ‘Output-Driven’ vs. ‘Outcome-Driven’ development).

5.3 Research Method

In this section, we describe our research method. This case study [137] builds on an ongoing work with five case companies and it was conducted between January 2015 and June 2017. The goal of this study was to identify a state-of-the-art differentiation of software features. Companies A-C were participating for the full period of the ongoing research while companies F and D joined in this research in April 2016.

5.3.1 Data Collection

We collected our data using three types of data collection activities. First, we conducted **individual workshops** with the companies involved in this research. Second, we conducted **joint workshops** with participants from multiple companies discussing the focus of our research. Finally, we conducted semi structured **interviews** following a guide with pre-defined open-ended questions with participants from each of the companies for in-depth analysis. During the group workshops, we were always three researchers sharing the responsibility of asking questions and facilitating the group discussion. The data collection activities were conducted in English and lasted on average three hours (individual workshops and joint workshops), and one hour (interviews). Each of the data-collection activities was conducted with practitioners in different roles (software engineers, project/product managers, sales specialists, and line/portfolio managers). In total, we performed 14 individual workshops, 7 joint workshops and 45 interviews.

5.3.2 Data Analysis

During analysis, the workshop notes, interview transcriptions and graphical illustrations were used when coding the data. The data collected were analyzed following the conventional qualitative content analysis approach [161]. We read raw data word by word to

derive codes. In this process, we first highlighted individual phrases that captured our attention in the transcriptions. We used color coding to highlight different topics that emerged during the analysis process. In this process, we reflected on the highlighted text several times and took notes. As this process continued for a few iterations (e.g. by analyzing the first few workshop notes and interviews), codes emerged from the highlights. In the next step, we sorted these codes into categories (e.g. grouping them by color). This type of design is appropriate when striving to describe a phenomenon where existing theory or research literature is limited. After we emerged with a few codes, we created definitions for those codes and continued to code the rest of the data using the definitions.

5.3.3 Threats to Validity

Internal Validity. As the participants were familiar with the research topic and expectations between the researchers and participants were well aligned, this can be a potential source of bias and thus a possible threat to internal validity. We mitigate this threat by providing unbiased overviews of the research area at the beginning of every data collection session, and avoiding suggestive interrogation during the workshops and leading questions during the interviews.

Construct validity. To improve the study's construct validity, we used multiple techniques of data collection (e.g. workshops, interviews) and multiple sources of data collection (product managers, software engineers, sales representatives, managers, etc.). The complete meeting minutes from the workshops and interview transcriptions were independently assessed by two of the researchers. This process was overseen by the third researcher for quality control.

External validity. Our results cannot directly translate to other companies. However, we believe that certain software companies could benefit from studying our results. Specifically, the results of

this research are applicable for companies that are transitioning from mechanical or electrical towards software companies, which may experience similar issues in differentiating and prioritizing software features.

5.4 Empirical Findings

In this section, we present our empirical findings. First, and to understand the current development practices in our case companies to identify the type and the extent of the feature that they are developing, we outline the current customer data collection practices in the case companies. We recognize that the practices to collect customer data do not differ with respect to the feature being developed, but rather depend on the perceived need of information ad-hoc. Next, we identify the challenges that are associated with distinguishing between the different feature types. Finally, we explore their implications.

5.4.1 Current State of Feature Differentiation

In the case companies, products and features that are being developed are handed over from one development stage to another, together with their requirements and priorities. Although our case companies acknowledge that they are aware of the need for differentiating between the types of features, i.e. commodity, differentiating and innovative, the information that would help them achieve that does not circulate with the requirements. The differentiation strategy is unclear to the practitioners developing the features. And consequently, setting the development investment level for a feature does not vary with respect to the type of the feature, allowing the prioritization strategy to be in favor of commodity features. We illustrate and describe the current state with description and quotes from our case companies below.

"Should we go into Maintenance budget? Or should it go to investment budget and we prioritize there?"

- Product Strategist from Company B.

"There is a lot of functionality that we probably would not need to focus on."

- Technology Specialist from Company B.

"Customer could be more involved in prioritization that we do in pre-development. Is this feature more important than the other one?"

- Product Owner from Company A.

5.4.2 Challenges with Feature Differentiation.

The current state advocates a situation where features are not differentiated in the strategy between being innovative, differentiating or commodity and, consequently, development activities do not differ between the features. Based on our interviews, we see that there are several challenges associated with this situation. Our interviewees report that the collection of feedback, which serves as the base for developing a feature is conducted ad-hoc and not following the type of the feature being developed. Second, the directives to differentiate the feature on a granular level are incomprehensible for our practitioners. Therefore, it is very challenging for the R&D staff in our case companies to be effective in understanding which of the features are differentiating and innovative in the market.

“We want to understand what the customer wants to have and truly, what do they need.”

- Product Manager from company C.

“Functionality is software basically and the features are more subjective opinions and things that we can’t really... it is hard to collect data.

-Function owner from Company B.

“Those are the things that worry us the most. All of us, since it is so hard, you need to gamble a bit. If it turns out that you are wrong, then you are behind.”

-Product Manager from Company C.

5.4.3 Implications

Due to an unclear differentiating strategy, our case companies experience several implications during the development process of a feature. The companies consider the stakeholders that the features are being developed for as a uniform group and not differentiating the development extent of the features based on the type of the feature and stakeholder. This leads to a situation where resources are used in developing and optimizing the features that might never get used by a customer to a similar extent as the ones that are regularly utilized. Consequently, commodity suppresses resources that could be better used to develop features in the innovation layer.

“If you are sitting in a team and then you talk to maybe 2 or 3 customers, then you see that this is the most important thing we need to do.”

-Product Manager from company C.

“We tend to focus on the wrong things. We need to look at the benefit for their customers.”

“In our organization is difficult to sew everything together, to make it work. That requires funding that is almost non-existent.”

-Software engineer from Company C.

“We do also our own tests of competitors. We do feature roadmaps for [feature name] for example. And then we have a leading curve... We measure are we on track or not.”

-Product Strategist from Company B.

5.4.4 Summary of our empirical findings

Vague differentiating strategy results in challenges with the purpose of the feature being developed, implicating uniform treatment of features. Invariable investment levels into development activities results in incomprehensible directives, implicating arbitrary investments in development activities. Favoring commodity results in a challenge of identifying what innovation, implicating the suppression of the actual innovation and projecting competitors current state as the norm. We summarize them in Table 7.

Current state	Challenges	Implications
Vague differentiating strategy	Understanding the stakeholder and purpose of the feature	Stakeholders treated uniformly, not reflecting their differing business value
Invariable investment levels	Incomprehensible high-level directives	Arbitrary investments in development activities
Feature prioritization processes is in favor of commodity	Commodity functionality is internally considered to be innovative	Commodity suppresses innovation Projecting competitors current state is the norm

Table 7. Mapping of the current state, challenges and implications.

5.5 Differentiating Feature Realization

In this section, and as a response to the empirical data from our case companies, we present and detail our model for feature differentiation. The contribution of our model is threefold. First, it provides four different categories of features and their characteristics to give practitioners an ability to better differentiate between feature types. Second, and as a guide for practitioners after classifying a feature, we provide a summary of development activities that should be prioritized for every type of feature.

By going through the process of articulating different views on the characteristics below, product teams can make the most of their collective insight on classifying them into “**duty**”, “**wow**”, “**checkbox**” or “**flow**” type. What is important for the practitioners, however, is how to optimally prioritize the development activities for each of the features types. We present this in the next section.

5.5.1 Feature Differentiation

In our model (visible in Table 8), four fundamentally different types of features are being developed.

Stakeholder	We recognize four types of fundamentally different stakeholders that are targeted with new feature development. Product users, the competitor developing or already selling a similar feature, the customer purchasing and asking for a feature, or a regulatory entity imposing it.
Feature Engagement	This characteristic describes the level of engagement expected with the feature. Features are primarily used by the end-users and occasionally by the customers directly. The engagement with the features is therefore expected for these two groups. Regulators and competitors, however, typically do not use the features directly. Instead, and what we see in our case companies is that verify the documents or tests demonstrating the existence or compliance of the feature. The expected exposure to the feature for regulators and competitors is therefore low.
Feedback	Feedback data that is collected about features under development is of various types. For example, the “Flow” features’ will be regularly used by users and should be equipped with automatic feedback collection mechanisms to retrieve customer data about feature usage. The “Checkbox” features source are the competitors that are being analysed for a similar offering. In the case of “Wow” features, the primary stakeholders are customers, whom companies study extensively through market analysis and available reports. Finally, and in the case of the regulation and standardization services (A), companies query regulation agencies for regulatory requirements.
Focus	“Flow” features focus on the user and maximizing user value. Practitioners, in this case, develop and iterate features that are validated with the users. For “Wow” features, practitioners use their own technical ability to maximize the technical outcome of the feature. Here, for example, companies use their simulators to test the speed of the software or its stability. For “Checkbox” features, practitioners compare the feature under development towards the ones from the competitors. In this case, the objective of the development organization is to develop a feature that is matching or improving the competitors’ feature. Companies continuously compare towards competition and intentionally slows down if they are performing better as expected.
Sales Impact	The greatest influence on driving sales have the features that focus on the customer “Wow” features and features that focus on the user “Flow” features. Both types are differentiating or innovative. However, in B2B markets, the user of the product is different from the customer and hence “Flow” features are less of a deciding factor for potential customers. As an example, users communicate and show satisfaction with a product to their managers and departments that purchase the features and products. Their opinions have the possibility to indirectly influence the opinion of the customer. “Duty” and “Checkbox” features have very low or no impact.

Table 8. Feature Differentiation.

5.5.2 Activity Prioritization

In this his section, we present the activities that should be prioritized for each of the four feature types. We suggest how to set the extent of the feature that should be developed. Here, the extent of the feature can be either defined once (constant) or dynamically adjusted during development and operation (*floating* alternates, *following* follows the competitors *or open* no limitation). Second, the sources that contain the information required to set the development extent need to be defined. Next, we suggest the most important activities for feature realization. They are followed by the activities that do not deliver value and should be avoided. Finally, we suggest how to set the deployment frequency. For details see Table 9.

Duty Features	The development extent for this type of features is constant and defined by the regulators. To identify it, practitioners can use research institutions, standardization industries and industry publishers as a source of feedback to get access to various standardization reports, vendor communications and obtain the regulatory requirements. For this type of features, identifying regulatory requirements and developing them until they satisfy the requirements are the two main activities. UX optimization, investments in marketing activities developing infrastructure and other similar activities for this type of features should be minimized. Deployment of this type of features is single.
Wow Features	For this type of feature, development extent is dynamically set using the feedback from the market. Practitioners query social media, marketing agencies, customer reports, requests, and interviews to identify business cases and sentimental opinions about the product. The two most important activities of this type are the identification of technical selling points and selling characteristic of the product, and maximizing investments in the technical development of the feature and marketing it. The feature's quantifiable value should be maximized with periodically scheduled deployment increments.
Checkbox Features	This type of features should be developed following a “sliding bar” extent that follows the competitor’s trend. Practitioners should guarantee to be on par with the indicative trend collected from industry test magazines and internal evaluation of customer products. It is essential to read articles in the media, consolidative question the customers, participate in customer events and perform trend analysis. Practitioners should perform competitor analysis to determine feature characteristics and develop the feature incrementally following the trend. Since features of this type will not be used extensively by the actual users, investments in improving user experience and interaction with the feature can be minimized. Deployment should be frequent and scheduled. Although products typically require this type of features to be even considered by the customers, they are not the decisive reason for customers to churn or select a product over a competitor.
Flow Features	With this type of features and the ambition to discover new and innovative concepts, practitioners should continuously deploy changes to their products, collect product feedback, analyse it, and perform A/B test to rank alternatives. The two most important activities for this type of feature are the defining of evaluation criteria and maximizing the ability to experiment with them. Interviewing the stakeholders or gathering qualitative information should be of secondary value and used for interpreting results. Also, and due to the continuous deployment, there is a high impact on the infrastructure.

Table 9. Activity prioritization.

5.6 Model Evaluation

We detailed our model and evaluated its feasibility with the five case companies in recent workshop and interview sessions. Our validation criteria were to identify whether companies identify the feature types that they develop (based on our model), and whether it helps them mitigate any challenges. Based on the evaluation, we identified that four of the five companies develop all types of features, whereas company E does not develop “duty” features. They do, however, acknowledge that they developed duty features in their early stages of building the key products.

Effortless Differentiation: One of the key challenges for companies is to identify whether a feature is a commodity, differentiating or innovative. With the feature differentiation model, practitioners felt empowered to perform this distinction based on the five key characteristics. By passing through the characteristics, practitioners can determine in which of the four quadrants their feature under development belongs to. We illustrate this with the two quotes next.

“Now we talk about these differentiators. This is known now, the wow features... One of the wow features, that we are selling is that we change rules and we demonstrate it in front of the customers.”

– Product Owner from Company B

“We have a lot of checkbox features in our products that are only there because the competitors have them, but it is not directly the end user that wants them.”

- Product Manager from Company D

Directive Comprehension: With the differentiation between the features ('Duty', Checkbox', Wow and 'Flow') and the development process (Output-Driven vs. Outcome-Driven), we give practitioners in large software companies the ability to define the right ambition level for a certain type of the feature, the preferred methods of collecting customer feedback, and provide them with instructions on which development activities to focus on.

"More experimentation is done in outcome-driven development. Absolutely in the Flow type."

- Product Manager from Company D

"For Duty and Checkbox features you know what you need to do, and you just have to do enough. You need to say that you have it, but it does not need to work that nicely."

- Product Manager from Company D

Distinguishing Innovative Features: With a clear separation between different types of features, our model enables practitioners to prioritize innovative functionality and invest in relevant activities, e.g. running continuous controlled experiments with customers for "Flow" features, or prioritizing investments into identifying regulation requirements for "Duty" features.

"You define the outcomes and see what activities contribute to that."

- Product Owner from Company B

5.7 Output-Driven vs. Outcome-Driven Development

In the previous sections, we provided guidance on how to differentiate between the different types of features and which development activities to prioritize for each of them. In this section, we illustrate the differences in the development approaches (based on additional learnings during the validation sessions) for the three stages of feature development (Value Identification, Value Realization, and Value Validation). For each of the stages, we provide indications on what is beneficial and a drawback for this type of development. For example, in the Value Identification phase (illustrated with 1 in Fig. 1), we discuss requirements freedom (the extent to which feature teams can interpret and change requirements) of both approaches. Next, we contrast ‘Value Realization’ phase (illustrated with 2 in Figure 4) by briefly presenting the differences in the autonomy of the development teams. In the Value Validation phase (illustrated on Figure 4 below), we compare the extent to which a completion goal is known to a development team, and how distant they are from the customer data (e.g. feedback that can be used to validate how well the feature satisfies the objectives).

Figure 4. Feature development in Output-Driven and Outcome-Driven development.

5.7.1 Output-Driven Development

Our findings indicate that ‘Duty’ and ‘Checkbox’ features development follows a common process. We select this label to emphasize the long-lasting tradition that our case companies followed in their transition from electronics to software companies. Companies that develop features in this way typically spend a considerable amount of time studying the requirements and/or competitors to determine what to develop in the Value Identification phase. Because of known policies, defined standards or competitor’s development outcomes (for example, a feature that was realized), they fix the requirements on what they should develop and information to which extent is given (for example, a safety feature should activate under 10ms). This enables teams in the Value Realization phase to focus on meeting the requirements and following the standards and sparing the team from customer representative roles.

The teams that develop these types of features have limited autonomy and they do not re-prioritize development activities at execution. This is not necessarily a drawback. Certain approaches for feature development [162] reject the autonomy of teams and consider it an overhead. Minimizing the number and significance of the prioritization decisions that the development teams must do makes teams more focused on the development activity at hand. Generally, the development team should be deciding on feature priority only after the priorities of regulations, the business, and the customer have already been addressed. Because of the Stakeholder objectives, the teams in the ‘Value Validation’ phase can benefit from knowing what exactly satisfies the objectives.

The resulting feedback used for evaluation, however, can be very distant. As described in the previous section, features developed in this way typically do not get instrumented with real-time feedback collection. This implies that development teams depend on the time distance of feature integration and deployment to the customer, and the time distance of feedback collection, which can be (and in several of features developed by our case companies is) significant. In this development process, our case companies define the following question and strive to answer it: “Are the customers using the

system and how?” and answer it using typically qualitative feedback such as observations and interviews, and quantitative raw logs.

5.7.2 Outcome-Driven Development

We contrast the approach above by presenting how ‘Wow’ and ‘Flow’ features are being developed. In what we label ‘Outcome-driven development’ development teams work significantly more iteratively and autonomously. In the Value Identification phase, and due to the nature of ‘Wow’ and ‘Flow’ features, teams invest into identifying what customers and users expect as an outcome and outline a feature idea around this.

As a result of studying the value as it will be experienced by the customer and not a requirement, it is very challenging to quantitatively describe the extent to which the feature should be developed. In the Value Realization phase, agile teams embed a customer representative within the development team and the customer representative determines the priorities for development. Therefore, the team has a high autonomy and ownership of the features that they are developing. This is, in principal different than traditional agile teams which follow the backlog items as defined and prioritized by a product owner. This is possible as the Value Validation phase is closely connected with the first two phases due to proximity to customer data.

In this development process, our case companies define the following question that they try to answer: “Are the customers efficiently achieving desired outcomes with minimal blocking?” and measure their success with prescriptive analytics (for example experimentation) on customer value metrics (for example task success, time needed to result).

5.8 Conclusions

In this chapter, based on case study research in five large software-intensive companies, we identify that companies struggle to differentiate between different types of features, i.e. they don't know what is innovation, differentiation or commodity, which is the main problem that causes poor allocation of R&D efforts and suppresses innovation. To address this challenge, we developed and detailed a model in which we depict the activities for differentiating and working with different types of features and stakeholders. Also, we evaluated the model with our case companies.

With our model, which differs from existing models and similar models (e.g. the Kano model [86]) in that it focuses on software products with rapid customer feedback capabilities, practitioners can (1) categorize the features that are under development into one of the four types and invest into activities that are relevant for that type, (2) maximize the resource allocation for innovative features that will deliver the most value, and (3) mitigate certain challenges related to feature differentiation.

Our model, however, still requires an in-depth validation on a larger scale to claim its general applicability. The current evaluation is based on qualitative impressions of the practitioners from our study, which is certainly a limitation. In future work, we plan to expand this model by studying how mature online companies differentiate between the different types of features that they develop, how their activities are prioritized, and validate the model using quantitative metrics (e.g. counting the number of features of individual type in each of the case companies).

6 FEATURE LIFECYCLE

Abstract

With continuous deployment of software functionality, a constant flow of new features to products is enabled. Although new functionality has potential to deliver improvements and possibilities that were previously not available, it does not necessarily generate business value. On the contrary, with fast and increasing system complexity that is associated with high operational costs, more waste than value risks to be created. Validating how much value a feature actually delivers, project how this value will change over time, and know when to remove the feature from the product are the challenges large software companies increasingly experience today. We propose and study the concept of a software feature lifecycle from a value point of view, i.e. how companies track feature value throughout the feature lifecycle. The contribution of this chapter is a model that illustrates how to determine (1) when to add the feature to a product, (2) how to track and (3) project the value of the feature during the lifecycle, and how to (4) identify when a feature is obsolete and should be removed from the product.

This Chapter is based on

A. Fabijan, H. H. Olsson, H. H. Olsson, and J. Bosch, "Time to Say 'Good Bye': Feature Lifecycle," *in Proceedings*¹ of the 42th Euromicro Conf. on Software Engineering and Advanced App., SEAA'16, (Limassol, Cyprus, Aug. 2016), 9–16.

¹ © 2016 IEEE. With permission from the authors.

6.1 Introduction

Starting with agile development [12] and evolving towards Continuous Deployment [53], software companies seemingly improve existing functionality and deliver new features to products quicker than ever before. In this way, value is assumed to be generated to customers and users continuously with every update and deployment [53]. However, although companies are adopting techniques to validate the value of the features during development, shortly after and while they are being deployed [154], [163], [49], [46], [31], very little is known on how to track feature value over its lifetime and what actions to perform when a feature no longer adds value to the product.

As new software features are being introduced, existing products and features age and become over time less and less amenable to necessary adaptation and change [164], [165]. And by fulfilling the functional requirements of a new feature, certain costs are typically introduced to a system. Let it be higher maintenance cost or increased product complexity for the overall system, new functionality increases the complexity of managing and operating a product [159]. As a consequence, and what we explore in this chapter, is that the value of the feature changes over time and, eventually, becomes negative. In that case, it is more beneficial for a company to remove the feature from the system than to keep maintaining it.

Based on the case study research in three large software companies, we study how feature value can be continuously validated during its lifetime and which actions to perform when the value that it delivers reaches or falls under a certain threshold. We recognize that the value of the feature is not measured over time and that it is not unique for all stakeholders across the organization. The contribution of this chapter is a model that illustrates how to determine (1) when to add the feature to a product, (2) how to track and (3) project the value of the feature during its lifetime, and how to (4) identify when a feature no longer adds value to the product and how it should be removed.

The remainder of this chapter proceeds as follows. In section II we present the background on product lifecycle and prior research on feature value. In III, we describe our case companies and the research methodology that we applied in this chapter. In section IV, we present the findings of this study and their challenges, which we address in section V with the introduction of the feature lifecycle model. Finally, section VI discusses the conclusions and future work possibilities.

6.2 Background

In software development a feature is a component of additional functionality, additional to the core software. Typically, features are added incrementally, at various stages in the lifecycle, usually by different groups of developers [166].

Ongoing expansion or addition of new features in software products, known also as feature creep or ‘featuritis’ , is a well seen pattern in software companies today, [167] [104], [52]. As new software features are being introduced, existing products and features age and become over time less and less amenable to necessary adaptation and change [164], [165]. Norman [168] recognized that software complexity increases as the square of the new features being introduced. Therefore, and in order to control the evolvement of a product and manage its development, we investigate product lifecycle management research literature.

6.2.1 Product Lifecycle

Product Lifecycle Management is a business activity of managing, in the most effective way, the product all the way across its lifecycle [71]. It is a process that starts with the very first idea about the product and lasts all the way throughout development and until its retirement. Although it is mainly associated with engineering tasks, it also involves marketing, product portfolio management [169] and new product development [170] activities.

A traditional product lifecycle consist of four stages i.e. the introduction, growth, maturity and decline [171]. In the introduction stage, the companies seek to build product awareness and identify

the market and potential customers. The value hypotheses about the features are proposed and being tested with potential customers, primarily with more qualitative feedback collection techniques such as interviews, focus groups etc. [59].

During the growth, the companies strive for reaching a wider market share by expanding their products and introducing new functionality. Customer feedback is typically collected in prototyping sessions in which customers test the prototype, discuss it with the developers and user experience specialists. At the same time, the initial hypothesis about product value are updated with new information [59], [152].

When the product is in its mature stage, companies implement strategies to maintain the market share by e.g. promoting the differentiating functionality that attracts and locks the customers. And although the companies are aware that more than a half of the features that already exist in their products are never used [172], they increasingly invest into developing new functionality. This results in a situation where many features attract similar amount of attention and receives an equal amount of R&D investments. As a result, more and more legacy exists in products, and companies spend increasing amounts of resources for managing the product and maintenance [164], [165].

The decline stage is where the cost and interest of investing into functionality that is not appreciated by the customers is being paid. At this time, companies are aware that functionality in the products is costing more to be maintained as it delivers revenue and are forced to either reduce costs and continue to offer it, possibly to a loyal niche segment, or completely remove it from the portfolio [171], [169].

And although the companies are aware that some features, at some point in time, cost more than what they add in terms of value, they do not have the data to support this and to understand when this happens. Assessing of feature value is, so far, an ad-hoc process with limited traceability and poor tool support [36].

6.2.2 Feature Value

Value from the customer point of view is an increasingly studied concept in the literature [173], [174], [175]. Porter's [174] value chain framework, for example, analyzes value creation at the companies level that identifies the activities of the firm and then studies the economic implications of those activities. Hemilä et al. [175] divide the benefits of a product or a feature among four types, i.e. functional, economic, emotional and symbolic value.

Functional feature value focuses on the functionality and offerings. Traditionally, value proposition was based on the functional features of product or services. Here, the value for the customer is e.g. new feature in the product or an improvement of the existing solution. Economic feature value focuses on price. Here, customers are e.g. offered the product or service functionality under a lower price than before or compared to the competitor. Emotional feature value focuses on customer experience. The meanings of emotions and feelings have already been important in e.g. automotive industry [175]. Today, however, they are increasingly becoming important also in the service system value creation [175]. Symbolic feature value focuses on the meaning. In the last years, consumers increasingly prefer to purchase products that are produced from recyclable material or run on green energy.

To measure, assess and visualize how much of value a feature delivers over time, multiple approaches are seen in the literature. The single metric approach described by Roy [176] treats feature value as a weighted combination of objectives and is highly desired and recommended for situations with a clear strategy. A single metric forces tradeoffs to be made once for multiple goals and aligns the organization behind a clear objective [154]. On the other hand, and when there are multiple strategies in the company, Kaplan et al. [177] suggests a balanced scorecard approach. This approach is valuable when there are multiple objectives in the companies' strategy and need to be fulfilled independently [178]. These objectives can then be linked back to organizational strategies using, for example, GQM+ concepts [179].

In this chapter, however, we recognize that despite the awareness of the increasing system complexity, feature value is not being systematically and continuously validated. This results in a situation where features are never removed and start to have, after a certain time, a negative effect on the product. We propose a concept of ‘feature lifecycle’ to demonstrate how the value of a feature in connected products can be assessed over time in order to be more proactive in conducting feature management decisions. The contribution of this chapter is a model that illustrates how to determine to add the feature to a product, how to track and project the value of the feature during the lifecycle, and how to identify when a feature is obsolete and should be removed from the product.

6.3 Research Method

This research builds on an ongoing work with three case companies involved in large-scale development of software products and it was conducted between August 2015 and December 2015. For the purpose of this research, we selected the case study methodology as an empirical method [137]. Since research in software engineering is to a large extent a multidisciplinary area aimed to investigate how development and maintenance of software features is conducted from a value point of view, we found the case study approach appropriate for this research [137]. To objectively obtain the detailed information on how companies perceive feature lifecycles, track and validated feature value, a real world setting was required.

Based on experience from previous projects on how to advance in continuous validation of customer value [46], [163], we held three separate workshops with each of the companies individually. After the workshops, we performed an interview study in which we met with a total of 22 people from the companies involved in this study. We list the participants and their roles in Table on the next side.

<i>Company and their domain</i>	<i>Representatives</i>
Company A	1 Product Owner 1 Product Manager 2 System Managers 2 Sftw. Engineers 1 Release Manager 1 Area Prod. Mng.* 1 Lean Coach* 1 Section Mng.*
Company B	1 Product Owner 1 System Architect 1 UX Designer 1 Service Manager
Company C	1 Product Owner 2 Product Strategists 2 UX Managers 2 Function Owners 1 Feature Coord. 1 Sales Manager 2 Technology Spec.

Table 10. Description of the companies and representatives.

6.3.1 Data Collection

During the group workshops with the companies, we were always three researchers sharing the responsibility of asking questions and facilitating the group discussion. Notes were taken by two of the researches. After each of the workshops, the notes were consolidated and shared to the third researcher.

We held semi-structured group interviews with open-ended questions [137]. These questions were asked by one of the researcher while two of the researchers were taking notes. In addition to the workshops, we conducted 22 interviews that lasted one hour and were primarily recorded using an iPhone Memo recording app. Individual Interviews were conducted and transcribed by one of the researchers. In total, we collected 13 pages of workshop notes, 138

pages of interview transcriptions, and 9 graphical illustrations from the interviewees. All workshops and individual interviews were conducted in English and lasted three and one hour respectively.

6.3.2 Data Analysis

During analysis, the workshop notes, interview transcriptions and graphical illustrations were used when coding the data. The data collected were analyzed following the conventional qualitative content analysis approach [180] where we derived the codes directly from the text data. This type of design is appropriate when striving to describe a phenomenon where existing theory or research literature is limited. Two of the researchers first independently and then jointly analyzed the collected data and derived the final codes that were consolidated with the third and independent researcher who also participated at the workshops. For example, when the interviewee referred to value of the feature, we marked this text and labelled it with a code 'feature value'. Similarly, we labelled the remainder of our interview transcriptions. As soon as any questions or potential misunderstandings occurred, we verified the information with the other researcher and participating representatives from the companies.

6.3.3 Validity considerations

Construct validity: We conducted two types of interviews. First, the group semi-structured interviews at the workshops and second, individual interviews with representatives working in several different roles and companies. This enabled us to ask clarifying questions, prevent misinterpretations, and study the phenomena from different angles. We combined the workshop interviews data with individual interviews. Two researchers, guaranteeing inter-rater reliability, independently assessed the notes. And since this study builds on ongoing work on feature value tracking and validation throughout the product lifecycle, the overall expectations between the researchers and company representatives were aligned and well understood upfront.

External validity: The results of the validation cannot directly translate to other companies. However, and since these companies represent the current state of large-scale software development of embedded systems industry [46], [163] we believe that the results can be generalized to other large-scale software development companies.

6.4 Findings

In this section, we present our empirical findings. In accordance with our research interests, we first outline the current state of assessing feature value throughout the feature lifecycle in our three case companies. Second, we list the challenges that are associated with these practices. We support each of the sections with illustrative quotes from our case companies.

6.4.1 Current State

In the case companies the value of the features is being assessed unsystematically and ad-hoc. Consequently, features are added to the product when they are built and removed when the management assumes no one uses them.

Tracking of feature value: Our case companies collect customer feedback as distinct points in the development cycle. They aggregate these data to build an understanding of feature value at feature prioritization time and after it is deployed in the field. However, and although various roles possess the partial data about individual values of the feature, they do not consolidate this information or monitor how it changes over time. As a consequence, and since every team looks at a different factor of the feature, the value differs throughout the organization, teams and units.

An objective measure of the value is thus never calculated, leaving the organizations with several different understandings of feature value for every factor that they analyze it for.

“The problem is that I don’t really see all the counters explicit-

ly... I only see some specific parameters. It is not really all.”

– System Manager from Company A.

Adding features to products: Our case companies continuously develop new features for their products. To validate the prototypes and steer development, they extensively collect customer feedback from the customers, users of the product, and directly from the products. When the development is done, this success typically becomes celebrated with as early deployment to the customers as possible.

“When things are working good, we are having a launching customer.”

– System Architect from Company B

Feature value is assessed ad-hoc: Initially validated value of the feature after its completion of development and deployment is assumed to remain near constant throughout its lifecycle. The next time the value of the feature is assessed again is when a customer experiences a problem with the feature.

“Normally we take the results that we see if we really deviate in one area, we hear it from customers”

– Product Strategists from Company C.

Features are seldom removed from the products: Once the features are build and deployed to the customer, they typically stay in the product. This causes the companies an increase in complexity and expenses for operations of the product.

“We could remove features. To remove the complexity! We are always increasing the complexity and that is a big problem.”

– System Manager from Company A.

6.4.2 Challenges

Based on our interviews, we see that there are a number of challenges associated with not being able to continuously assess feature value throughout its lifecycle. For example, our interviewees all report difficulties in identifying the value of the feature and using this information to identify the next actions, i.e. when to deploy the functionality or remove it from the system.

Tracking and projecting feature value over time: Since there is no traceability between one measurement of feature value and the second one, knowing whether a new version of the feature is better or not is not intuitive. This situation is illustrated in the following quotes:

“We get feature which is broken down and then this value somehow got lost when it was broken down.”

– Software engineer from Company B.

“I would like to be able to provide the sales force with more description of the actual features products benefits services in a simple speech, but also the quantitative figures, in what situation, what segment...”

- Sales manager from Company C.

Identifying when it is valuable to deploy features: Although the principles of lean methodology stress early deployments and learnings, features should be deployed not when they are ready but when it is valuable to do so, i.e. when they will deliver value greater than a certain threshold defined by the strategy. Since the traceability of feature value is limited, features in our case companies are deployed even though they don't deliver enough value in that moment. We illustrate this situation in the following quote:

“It would be great if we could draw those possibilities on a whiteboard and get the percentage.”

- Product Manager from Company A

Identifying when to remove features from the product: Due to a lack of objectively assessing feature value and projecting it over time, it is very difficult for our case companies to know when the feature does not deliver any more value as it costs to have it in the product. Consequently, removing the features from the product is done unsystematically and with consequences. Moreover, our case companies report that features stay in the products even though they are aware that no one is using them, buying the product because of them, or serve any compliance purpose. This situation is illustrated in the following quotes:

“A developer wants to know if the code is being used so he knows if he or she can remove the code. How big the impact will be or what value should we set for that.

-Software Engineer from Company A

“Can we obsolete certain parts of the system that we maintain for backward compatibility reasons but maybe they are not used or maybe they can be replaced by something else?”

- Service Manager from Company B

“If we go 5 years back, we see that we have many features that have never been tested or used. It is a beauty contest.”

- Product Owner from Company A

6.5 Feature Lifecycle Model

To assess feature value and model how to track it over its lifecycle, we first provide a feature value equation. Based on a single metric approach that consolidates multiple factors, this equation captures quantitative value for features in their lifetime. Second, we present the general feature lifecycle model as a function of value equation over a typical lifetime of a feature.

6.5.1 Feature Value Equation

Typically, companies initiate new feature development with a strategy decision. At this time, high-level requirements yield an idea about the feature purpose and what the net benefits will be for the company when this particular feature is in the product.

We describe the calculation of feature value ‘ V ’ over time ‘ t ’ as a single metric approach and define it as a matrix product between value factors ‘ F ’, their relative weights ‘ W ’ and an interaction matrix ‘ I ’. We illustrate this with Equation 1 below.

$$V(t)_{i \times i} = (F(t)_{i \times j} \cdot W(t)_{j \times i}) \cdot I_{i \times i} \quad \text{Equation 1.}$$

The number of lines ‘ i ’ in factor Matrix ‘ F ’ equals to the number of features being assessed. Every line ‘ i ’ in matrix ‘ F ’ contains factors that relate to feature ‘ i ’. The number of columns ‘ j ’ equals to the number of these factors that constitute feature value. They originate from the Key Performance Indicators (KPIs) and high-level strategy guidelines that are available within the organization, and need to be normalized. Examples of these factors are capacity, system load, latency, net profit, market demand, feature usage, etc. Previous researchers abstract these factors and distinguish, as we describe above, between four different types: Functional, Economic, Emotional and Symbolic [175]. In our model, and in order to be aligned with the distinction, we recommend arranging related factors in matrix ‘ F ’ together (i.e. keeping columns that resemble functional value on the beginning of the matrix, followed by columns with factors that resemble economic, emotional and symbolic value).

Since not all of the factors are of equal importance, we weigh them with the corresponding column from matrix ‘ W ’. In each column in ‘ W ’, the weights represent the relative importance of each of the factors for a certain value. Although the weights can differ throughout the matrix ‘ W ’ (e.g. a certain factor can have more importance for one feature, and another factor for yet another feature), the weights are required to be allocated in a normalized fashion.

ion. Consequently, the sum of values in every column in matrix 'W' equals to one.

We name the third matrix to be the interaction matrix 'I', which takes into the consideration the interaction between new and existing features in the product under development. Features are typically developed and tested in isolation, or with a particular product use-case. However, when several features are added to a product, there may be interactions (i.e. behavioral modifications) between both the features already in the product, as well as with features in a related product. Adding a new feature which interacts with other, existing features implies more effort in the development and testing phases. And although the focus of this research is not to study how features interact with each other (for that see e.g. [166], [181]), we believe that the possibility to take the interaction into the calculation of the feature value should exist. When the interaction is non-existent, i.e. between features that do not change the behavior of each other, the interaction matrix is an identity matrix (ones in the diagonal and zeroes elsewhere). On the other hand, when the interaction is recognized, the relative contribution between pairs of features increases from zero to the percentage detected in the interaction.

We provide an example calculation of feature value using Equation 1 for three features that do not interact with each-other on Figure 5.

$$\underbrace{[V_1, V_2, V_3]}_{\text{Feature Values}} = \underbrace{\begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{i1} & F_{i2} & F_{ij} \end{bmatrix}}_{\text{Factors}} \cdot \underbrace{\begin{bmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \\ W_{i1} & W_{i2} & W_{ij} \end{bmatrix}}_{\text{Weights}} \cdot \underbrace{\begin{bmatrix} 1 & \dots & 0 \\ \vdots & 1 & \vdots \\ 0 & \dots & 1 \end{bmatrix}}_{\text{Interaction}}$$

Figure 5. An example calculation of feature value using Equation 1.

In this example, and by using standard matrix multiplication between the factor matrix ‘F’ weight matrix W, the factor line ‘F11, F12, F13’ is multiplied with its relative importance column ‘W11, W21, Wi3’. The result is a square matrix multiplied with an interaction matrix. The later in this particular case is an identity matrix as no interaction is assumed. The result is an array of values ‘V1, V2, V3’ for every feature. Should there be interaction between features, the final calculation would capture the proportion of the interaction in each of the values due to the matrix multiplication effect.

6.6 Feature Lifecycle Model

We focus on how value of a feature changes over time and what actions can be taken at certain milestones. We illustrate an example lifecycle of a typical software feature on Figure 6. On the horizontal axis we list the most important milestones in a feature lifecycle i.e. feature development start – t_0 , first deployment – t_1 , adoption – t_2 , peak – t_3 , equilibrium – t_4 , removal optimum – t_5 and finally, the removal of the feature – t_6 .

On the vertical axis, we plot feature value over its lifecycle. Although illustrated as a parabolic arc for a better presentation, dotted lines around the descending curve on Figure 6 should remind

Figure 6. Feature Lifecycle Model.

the reader that in practice the value does not necessary increase or descend as an exact parabolic function of time. Time segments between value changes can be shorter or longer, depending on practitioners development practices, feature feedback collection intensity etc. Consequently, the value of the function may alternate to a certain extent within the segments, both due to the development practices and natural variation of the collected data.

We describe the segments of the feature lifecycle model, which we label on Figure 6 with A-E, in the paragraphs below.

A – Feature Infancy: In the early phases of pre-development, companies start to construct an understanding of what will the value of the feature constitute of. We label the time t_0 to be the moment in time when feature first appears at the strategy level, where early hypotheses are being formed. During the period A, which lasts from the moment t_0 until the feature is deployed to the first customer t_1 , companies collect qualitative customer feedback using customer journeys, interviews, questionnaires and survey forms to hypothesize what their customers’ needs are. At this stage,

contextual information on the purpose of the feature with functional characteristics and means of use are typically collected by customer representatives. Typically, this information is used to form functional value requirements of the feature, which translate to factors ' F ' in Equation 1. At the same time as the factors are defined, weights in ' W ' are determined. For example, if the feature is facing the user, more weight might be assigned to usability, experience and similar factors. On the other hand, and if the feature being modeled is satisfying a regulation, usability and experience receive less weight, which is instead assigned to i.e. compliancy.

The development units in the company develop early prototypes of the feature and measure how close the realized product is in comparison to the requirements. Since no customer is actually using the feature in this period A, no value for the company is generated (illustrated with a flat line on Fig. 1).

B - Feature Deployment: When a minimal feature instance is developed to the extent that it can be validated, practitioners can deploy it to a test customer, and for the first time, assess how well their implementations realize the expectations in the field. Typically, at this moment companies would deploy the feature in a 'silent mode' to the customers to maintain integrity and activate it incrementally with the test customer. We label the point in time when the first real customers adopt the feature with $t1$. The feedback at this moment is typically mixed and needs to be quantified. Typically, it consists of both qualitative information on e.g. design decisions and quantitative operational data [59]. With new input, the values of the factors in matrix ' F ' are continuously updated using the quantified customer input. As a result, practitioners get feedback on product behaviors and initial performance data that can be used to calculate the current value of the feature.

C - Feature Inflation: After the first deployment $t1$, customer adoption of the feature intensifies. Companies strive to provide marginal improvements that will differentiate them from the competitors. For this purpose, they continuously experiment with the customers to identify the version of the feature that delivers the most value. New iterations of the feature are continuously de-

ployed and controlled A/B experiments can be performed to identify the gain [59], [40]. Due to the approach in Equation 1 with a single metric that forces tradeoffs to be made once for multiple goals, companies are enabled to determine whether the incremental improvement actually delivers more value compared to the version before [154]. Again, and with new input, the values of the factors in matrix F need to be continuously updated using the customer feedback. The feature development stops completely at the moment when there is no statistical difference in feature value between sequential iterative versions of the feature.

D – Feature Recession: Customers will typically stop paying for existing features in favor of something newer and better. At a certain point in time, the value of the function will reach the peak, which we label on Fig.1 as t_3 . Finding the maximum of the value function is the goal of mathematical optimization. And since feature value is calculated on an ongoing basis, companies can store the historical values and use them for projection trends. By identifying the equation of the curve using e.g. polynomial regression, practitioners can find the point when the value starts to steadily decline. The peak value t_3 is the point where the first derivative of the regression function equals to zero and the second derivative is negative. From hereafter, the directional coefficient of the first derivative is negative, projecting a decline of feature value.

There may be several reasons for the value of a function to start steadily decreasing. Although the aim of this research is to confirm this phenomenon and not to address its causes, we name a few of them. Features age and occasionally lose their purpose over time. Customers might churn to the competitors or use existing alternative functionality. Also, and since new features are increasingly being introduced to the product, existing features get replaced or even interact negatively with the new ones [181]. Consequently, they cause increase in operational expenses [164], [165].

As an example of reducing system complexity and consequently, lowering the costs of the operation, companies might consider to start commoditizing and opening the features to a wider communi-

ty [159]. By open-sourcing the features, resources are freed from maintenance and support. Consequently, and by lowering the operational costs, the derivative of the value function becomes less negative, which enables maintaining the value of the feature at a higher level for a longer time (the slope after t_3 in Fig 2 is decreasing at a slower pace).

E – Feature Removal: Regardless of the restructuring and efforts made into reducing feature costs, the value of the feature essentially becomes equal to the costs of its operation and maintenance. We label this point in time on Fig. 2 as t_4 - value equilibrium. Although not yet costing, product management should already consider ways of removing this feature from the product. Ultimately, product managers should seek to identify any feature whose removal would have no impact on the value of the overall product. When the monetary value of the feature falls below the maintaining and supporting cost, as the feature presents a liability to the product and detracts the customers from other value-adding features. At this time, and until point in time t_5 , the cost of removal of the feature is lower or equal to the operational costs of the feature. The point t_5 is the point in time where the cost of removal a feature is lower as the interest the company will pay by not removing the feature after this time.

6.7 Discussion

Ongoing expansion and addition of new software features in software products is a well seen pattern in software companies today [167] [104], [52]. With many new features being introduced on a continuous basis, existing features age and become over time less and less amenable to necessary adaptation and change [164], [165]. And although the companies are aware that some features, at some point in time, cost more than what they add in terms of value, they do not have the data to support this and to understand when this happens. Assessing feature value has, so far, been an ad-hoc process with limited traceability and poor tool support [36]. As a consequence, companies do not deploy features when they deliver value or remove them from the product when they stop being valuable.

In response to the challenges and implications presented above, we suggest the feature lifecycle model. There, we illustrate how feature value can be defined as a linear combination of factors and weights and validated over time. This helps companies in knowing when to deploy a feature or remove it from a product. In particular, the model helps companies with:

Validating feature value over time: The single metric approach in our value equation is a compact method of combining feature factors that constitute the value of the feature with their weights. In particular, this approach allows that there might be several variables that constitute feature value (factors), each with a different importance (weight). Moreover, the model combines in a single metric weights and factors for multiple features, taking into consideration the possibilities with alternating weights and feature interaction. As a result, the equation yields feature value as numbers which are effortless to compare in-between. This is especially useful in situations when smaller increments of the feature are being tested and product management requires fast-feedback whether the new increment actually delivers more value in comparison to the old versions of the feature.

Projecting feature value over time: By accumulating feature value that was calculated using our model, practitioners are enabled to use the data to project future trends. For this, several supervised machine learning algorithms can be used. Polynomial regression, which we mention in our chapter is one of them. Essentially, the projection of feature value can both help companies to better plan their resources for possible feature maintenance or removal, as well as to better estimate the future gain or loss of value.

Deploying features when it is valuable to do so: With a clear calculation of feature value using a single metric, companies can deploy features when they deliver the value, instead of when they are developed. Our model suggest the deployment in a ‘silent mode’ to allow for a timely validation with test customers and revealing the feature to other customers only when its value is above a threshold defined by the strategy. By projecting feature value over time, the practitioners are better aware how they are progressing and which factors require more attention.

Removing features from the product: The feature lifecycle model raises awareness about software features and changes the perception about their lifecycle. It illustrates that the value of the feature not only drops over time, but also becomes negative. The model helps practitioners to identify when the equilibrium will change to imbalance, costing the product and organization. In particular, it enables practitioners to plan the removal before the negative value of the feature costs more than the actual process of removing the feature.

6.8 Conclusions and Future Work

With continuous deployment of software features, a constant flow of new features to products is enabled. As a result, and with fast and increasing system complexity that is associated with high operational costs, more waste than value risks to be created over time.

To help companies assess feature value throughout the lifecycle, and to avoid situations with features in products with negative impact, we propose and study the concept of a software feature lifecycle from a value point of view. Based on the three case companies, we demonstrate how to track feature value throughout its lifecycle. The contribution of this chapter is a model that illustrates a feature lifecycle. It helps companies to determine (1) when to add a feature to the product, (2) how to track and (3) project the value of the feature during the lifecycle, and how to (4) identify when a feature is obsolete and should be removed from the product.

7 EXPERIMENT LIFECYCLE

Abstract

Online Controlled Experiments (OCEs) enable an accurate understanding of customer value and generate millions of dollars of additional revenue at Microsoft. Unlike other techniques for learning from customers, OCEs establish an accurate and causal relationship between a change and the impact observed. Although previous research describes technical and statistical dimensions, the key phases of online experimentation are not widely known, their impact and importance are obscure, and how to establish OCEs in an organization is underexplored. In this chapter, using a longitudinal in-depth case study, we address this gap by (1) presenting the experiment lifecycle, and (2) demonstrating with four example experiments their profound impact. We show that OCEs help optimize infrastructure needs and aid in project planning and measuring team efforts, in addition to their primary goal of accurately identifying what customers value. We conclude that product development should fully integrate the experiment lifecycle to benefit from the OCEs.

This Chapter is based on

*A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch, "The (Un)Surprising Impact of Online Controlled Experimentation," **in Revision** in an International Software Engineering Journal.*

7.1 Data-Driven Decision-Making

Companies have always been collecting data to understand what the customers value and make decisions based on the lessons learnt [38]. Although techniques for collecting customer feedback have traditionally focused on the collection of what the customers say they do (for example interviews or focus groups) [182], the software industry evolves towards complementing these with the collection of data that reveals what customers actually do [10], [183]. This enables software companies to be more accurate in evaluating their ideas, and to move away from assumptions and towards trustworthy data-driven decision making. The internet connectivity of software products provides an unprecedented opportunity to evaluate hypotheses in near real-time. Consequently, the number of hypotheses that product management generates and aims to evaluate can be tremendous. For example, there are billions of possibilities merely to style a product (e.g. ‘41 Shades of Blue’ OCE at Google [57]). In fact, style and content management is only one area for experimentation. Evaluating improvements to ranking algorithms and recommender systems [15], [184] are popular applications of online experimentation, among others. And although there are several ways to evaluate hypotheses (with e.g. pattern detection, classification, etc.) none of them shows as direct causal and accurate impact of the idea on the customer value as Online Controlled Experiments do [184].

7.2 Online Controlled Experiments

In an OCE, users of a software product are randomly distributed between several variants (e.g., different designs of a product interface) in a persistent manner. The existing variant without the change is labeled as “control”, whereas the other variants as “treatments”. The goal of this set-up is to accurately evaluate ideas about software products (e.g. whether changing the checkout button color from gray to blue increases the number of purchases). While the user interacts with the product, activity signals (e.g. clicks, dwell times, etc.) are instrumented and key metrics (e.g. en-

gement, user satisfaction, etc.) are computed [29], [183]. If the difference between the metric values for Treatment and Control is statistically significant, the introduced change very likely caused the observed effect on the metrics. With this, OCEs provide an accurate understanding of how much customer value the change in the products actually delivers.

7.3 The Impact of Experimentation

Incremental product improvements are not the only scenario in which OCEs impact product development. A number of benefits can be experienced as a result of OCEs [9], [25], [29]. This section, based on our analysis of several hundred experiments at Microsoft, summarizes those most commonly observed.

First, OCEs have an impact on how *work effort is* (I_1) ***planned and*** (I_2) ***measured*** for feature teams. Predicting impactful changes to the product is very difficult without OCEs. By identifying the changes that showed the most impact on the product-wide metrics, teams learn from each other and re-prioritize their work. At the same time, teams' contributions are measured not only by the number of changes they deployed but also with the exact amount of metrics' improvements. With OCEs, performance goals for teams take on a new meaning.

Second, OCEs impact how *product teams ensure* (I_3) ***incremental improvement of the product and*** (I_4) ***product quality***. By only deploying the changes that increase customer value, and since many quality issues can only be detected while the products are used by the customers, software products are becoming better in every sense. Also, OCEs enable companies to (I_5) ***optimize infrastructure needs***. By observing the effect of a treatment on a subset of users while in the Design & Execution phase, infrastructure capacity can be estimated for the full deployment.

Finally, OCEs enable an accurate (I_6) ***discovery and validation of customer value***. Since they control for all other factors, they enable to accurately identify what the customers value, and which changes contribute to that. Knowing this is not only important for

the feature teams developing the features, but also for other product-wide teams that are planning investments (see e.g. (I_1)).

“We are often surprised by the impact of a change which we thought would be inconsequential, like changing the background color or adding a line on a web page, on the overall revenue.”

- Data Scientist

7.4 Experiment Lifecycle

To derive the full impact of OCEs and understand how they enable large-scale companies do trustworthy decision making, we conducted a longitudinal in-depth case study [138] at Microsoft’s Analysis & Experimentation group. We analyzed hundreds of experiments conducted at Microsoft between January 2015 and September 2017. Based on this study, which involved the review of data from eight different product teams, we present the experimentation lifecycle.

In many teams, OCEs at Microsoft are an integral step of the feature release process handled by Experiment Owners (EOs). Experiment Owners execute experiments to evaluate their ideas and are responsible for them throughout their lifecycle. An experiment lifecycle consists of three phases: (A) - Ideation, (B) - Design & Execution, and (C) - Analysis & Learning. The focus of the Ideation phase is twofold: (1) Generating a Minimal Viable Hypothesis (MVH), and (2) developing a Minimal Viable Product (MVP). The Design & Execution phase is concerned with the (1) creation and (2) execution of the experiments. Finally, in the Analysis & Learning phase experiments are (1) evaluated and the learnings from them are (2) institutionalized. A thorough understanding of the phases and their integration in the product development process is critical to fully benefit from OCEs.

On Figure 7, we illustrate this partition and discuss each of the phases in greater detail next.

Figure 7. Experiment Lifecycle: (A) - Ideation, (B) –Design & Execution, and (C) - Analysis & Learning

7.4.1 Ideation

Ideas for OCEs are inspired from past experiments, feature teams, management, etc. EOs **devise** and **disseminate** them, analyze the feedback on their development complexity, and decide whether and how to form Minimal Viable Hypothesis (MVH). An MVH consists of a **treatment description** (e.g. “changing the background from white to gray”) and **success criteria** (e.g. “increases user engagement by 1%”).

The MVP is the **implementation** of the hypothesis in the product by focusing on the MVH ‘treatment description’ attribute. What distinguishes an MVP from a final product is not the lack of testing effort, but the difference in the implementation. For example, since the treatment variant will be initially exposed to a smaller subset of users, it does not need to scale in the same way as its final version.

7.4.2 Design & Execution

EOs design the experiment by specifying the **customer segment** in which the hypothesis will be evaluated (e.g. ‘the US’) and conducting the **power calculation** to determine the size and duration of the experiment (see e.g. [185]). Typically, they also define the targeted experi-

ment start date, number of variants, randomization units (e.g. users or devices), and design plan (e.g. A/B, multivariate, etc.).

Once the experiment starts, EOs monitor it to detect large degradations. Automated alerts on key metrics can be configured for this purpose. With a pervasive democratization of experimentation towards EOs without statistical background, new challenges emerge.

7.4.3 Analysis & Learning

When the statistical power of the MVH is fulfilled, the experiment is analyzed. The **metrics are computed** and analyzed for statistically significant changes. Specifically, success criteria of the MVH are compared to the actual experiment outcome, and a **decision** on whether the experiment was successful and which variant won is made.

The impact of experimentation should extend **beyond** the decision on which variant of the experiment to deploy. By learning which type of changes influence the key success metrics and by how much, the **prioritization** in the Ideation phase is reconsidered and the cycle restarts with the next experiment in the queue based on the learnings obtained.

7.4.4 Challenges & Mitigations

First, MVH should be placed into a queue with the other hypotheses to get **prioritized** based on the expected impact and weighted against an estimated cost of developing. Such prioritization is more accurate if a **relation** between the effect of the experiment and product-wide metrics can be predicted (see ‘Institutionalization’ below).

Next, MVP should be tested for stability and common issues that could cause bad user experience. It can be challenging to know **when** an MVP is ready for experimentation. Previous work addresses this by testing an MVP on **smaller sample** and gradually increasing the treatment group (e.g. from minor to major markets) to get more sensitive results [9].

Concurrent experiments might interfere with each other (e.g. experimenting with text and background colors may result in devas-

tating outcomes) and such interferences should be automatically detected and prevented. Related work discusses **scheduling** or **isolation groups** to overcome this challenge [9], [57].

Eventually, EO's need a **platform** that helps them initiate, **iterate**, and analyze experiments. Open-source or commercially available platforms such as Wasabi or Optimizely can be utilized [25]. Over time, an in-house platform with advanced features may be needed [9].

Identifying **experiment insights** without tools that automatically find and highlight them can be challenging. Experiment results for in one user segment (e.g. the EU market) might differ the aggregated world-wide results. More research is needed in this area to accelerate the analysis of OCEs.

Experimentation at scale should enable EOs to step back and question the underlying assumptions about and **relations** between the low-level metrics that their experiments are impacting, and the higher-level product-wide metrics that are difficult to move in a short-term experiment [15], [25], leading to better metrics. Although previous attempts have been made to automate the institutionalization of experiment learnings [9], more work is needed.

7.4.5 Example Experiments

To demonstrate the efficacy of OCEs in relation to the presented impacts (I_1) - (I_6), we select four example experiments and present them in Table 2. We visualize the ones with the user interface on Figure 8 below.

Figure 8. Office Word App OCE (A), Birthday Greeting OCE (B), and MSN Video OCE (C).

We discuss each of the experiments in relation to our Experiment Lifecycle next.

7.4.5.1 Office Word App OCE

1. **Context.** Office mobile apps product team conducted an OCE with their Word mobile app.
2. **Ideation.** MVP: Office team developed and introduced the contextual command bar (see Figure 8 (A)). MVH: App users will do more editing on the phone because the new command bar will improve editing efficiency. Consequently, this will result in increased commonality and frequency of edits, and increased user retention.
3. **Design & Execution.** An OCE was designed with one treatment (new bar) and one control, and sufficient power to detect 1% changes in metrics. The OCE was executed once for 2 weeks.
4. **Analysis & Learning.** The results of the OCE indicated a substantial increase in engagement (counts of file edits), but no change in user retention. The learnings were shared with other teams. The experiment was considered successful. In addition, a need to conduct more experiments of this type to understand the relationship between editing activity and user retention was identified.
5. **Impact.** (I1) *Work effort measurement:* The team identified the exact impact of introducing this type of design change to one of their products on key metric. (I2) *Work effort planning:* These learnings were strong evidence to generalize the knowledge and confidently prioritize similar experiments in other related products.

7.4.5.2 Birthday Greeting OCE

1. **Context.** Bing product team evaluated the impact of a feature that displays a birthday greeting.
2. **Ideation.** MVP: For registered users on the day of their birthday anniversary they would display the animated birthday greeting (see Figure 8 (B)). The existing variant did not see this message. VH: The Bing team hypothesized that this feature would increase usage of Bing and will not introduce harm.
3. **Design & Execution.** The OCE was designed with a control without - and a treatment variant with the birthday feature. Due to initial findings (see below), the experiment was iterated.
4. **Analysis & Learning.** Several % of users that saw the promotion, engaged with it. Offline data, however, showed abnormally high number of birthdays on January 1st, likely most of which are not true. A follow-up iteration increased the prominence of “Update your Birthday” link for January 1st birthdays. About 1/5 of users who received this birthday wish initiated the update.
5. **Impact.** (I3) *Incremental improvement*: The team deployed the feature confident that it improves their product. (I6) *Discovery of customer value*: Entering the correct birthday is a clear signal of a stronger commitment to use the product, but it wasn’t part of the Bing key product-wide metrics before.

7.4.5.3 Bot Detection OCE

1. **Context.** Finding search results is a computational operation that consumes resources.
2. **Ideation.** MVP: Bing product team developed a more pervasive bot-detection algorithm. MVH: The Bing product team hypothesized that with an improved algorithm, (1) users will not be harmed and (2) fewer resources will be used for the computation of results.
3. **Design & Execution.** An OCE was designed with one treatment and one control. The OEC was iterated several times to tune the bot detection algorithm.
4. **Analysis & Learning.** Initially, experiment results showed degradations in key metrics. That indicated that real users were being harmed. After several iterations, no user harm was detected while much larger number of bots were identified.
5. **Impact.** (15)*Optimizing infrastructure needs:* The results of the OCE indicated that by excluding any other factor beside the change in the prediction algorithm, a ~10% saving on infrastructure resources is achieved.

7.4.5.4 MSN Video OCE

1. **Context.** MSN evaluated the value of showing immersive video experience.
2. **Ideation.** MVP: MSN team created a version of the video player that covers a larger area of the page (see Figure 2 (C)). MVH: Team hypothesized that the larger video player enables a more distraction-free experience.
3. **Design & Execution.** An OCE was designed with one treatment and one control. Experiment was iterated two times.
4. **Analysis & Learning.** The first iteration of the experiment caused an alarm indicating a difference between the observed and the expected split of users between the variants. The analysis of the experiment data revealed a bug with the product's configuration files.
5. **Impact.** (I4) Product Quality. The MSN team learned through an experiment that a previously unknown bug exists in their product. By removing the bug, they improved the quality of their product.

7.5 Integration Yields Impact

At Microsoft, OCEs are critical for consistent data-driven decision making. Their full impact, however, has become realized only when the phases of the experiment lifecycle became an integral part of the software development process [10]. For this to happen, the traditional "plan-build-test-deploy" process evolved into a data-first development process, where decisions for a product, infrastructure and the team are based on experiment evidence. Integration of the experiment lifecycle progressed gradually through a four-step evolution of the technical, business and cultural evolutions described in [10], starting within a team of skilled data scientists and pervasively expanding throughout the organization towards non-experts. Although the experiment lifecycle partition might seem intuitive when considering a single experiment, a thorough understanding and its alignment with product development cycles is crucial. For example, determining the final release date for

a feature or product is more challenging when doing OCEs, since multiple iterations may be required, and the process needs to accommodate this. Investment in talent and infrastructure to conduct OCEs is needed as well, though for companies without internal expertise commercial and open source platforms are available to implement parts of the lifecycle. Once integrated into the development process, OCEs provide an unprecedented advantage that is difficult to sidestep and quickly become business critical. The gain of being able to continuously and accurately understand which ideas actually increase customer (and consequently business) value provides handsome return on the investment needed for the lifecycle to be integrated.

7.6 Conclusion

The fact that OCEs enable software companies to accurately and continuously evaluate ideas with customers in product development activities is not novel [4], [29]. What is surprising, however, is that the impact of experimentation extends beyond the traditional use in deciding whether to deploy a change or not. Our learnings show that software companies can efficiently learn more and more with every experiment. OCEs profoundly change how team effort is planned and prioritized, and how infrastructure capacity needs can be optimized and predicted. To benefit from the full impact, however, the phases of the experiment lifecycle need to be understood and fully integrated into the product development process. Despite many challenges with integrating the experiment lifecycle, mitigations have been made available for many of them. For the Big Rocks that still stand on our way, such as the automation of OCE analysis and institutionalization of learnings, more research is needed.

8 EVOLVING EXPERIMENTATION

Abstract

Companies need to know how much value their ideas deliver to customers. One of the most powerful ways to accurately measure this is by conducting Online Controlled Experiments (OCEs). Microsoft, Google, Amazon, and many other companies are able to conduct thousands of OCEs per year engaging millions of users with different experiences. For product teams to be able to experiment at this velocity, however, companies need to develop strong experimentation practices as well as align their organization and culture to experimentation. The main objective of this chapter is to teach companies how to run OCEs at large scale using the experience of companies that succeeded in scaling. Specifically, we generalize our findings from prior research and provide detailed prescriptive guidelines that help to evolve experimentation capabilities. Based on an in-depth case study at Microsoft and three other software companies, we present our main contribution - The Experiment Growth Model. This four-stage model addresses the seven most critical aspects of experimentation. Companies can use the Experimentation Growth Model to evaluate the state of their maturity for experimentation.

This Chapter is based on

A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch, “The Evolution of Continuous Experimentation in Software Product Development,” in **Proceedings**¹ of the 39th International Conference on Software Engineering, ICSE’17, (Buenos Aires, Argentina, May 2017), 770–780.

A. Fabijan, P. Dmitriev, C. McFarland, L. Vermeer, H. H. Olsson, and J. Bosch, “The Experimentation Growth: Evolving Trustworthy A/B Testing Capabilities in Online Software Companies,” **In Revision** in an International Software Engineering Journal.

¹ © 2017 IEEE. With permission from the authors.

8.1 Introduction

Companies are successful only if they truly understand their customers' needs, and develop products or services that deliver value to fulfil them [6], [28]–[31]. The number of new features, product improvements, or other ideas that software companies in particular generate and hope they will impress the customers, however, can be tremendous [157]. And although some of them deliver as expected, most of the changes to the products either have no effect, or hurt the customers [186]. At Microsoft, only one third of ideas deliver the desired positive impact [11], [29]. Therefore, and to know where to invest, it is critical for companies to accurately measure how their customers react on the changes and features that they add and do to the products, and accurately interpret whether they are positive or negative.

Context. To understand what the customers say they value and make decisions based on the lessons learnt, companies have always been collecting feedback data [38]–[41]. Traditionally, feedback collection techniques (for example interviews, focus groups, observational studies, etc.) have focused on asking the customers what they say they do with the product, and what they *say they like or dislike* with it [70], [182]. Today, however, the software industry is evolving towards complementing their learning process about the customers' preferences with the collection of data that reveals what their customers *actually do* with the product [10], [59], [183]. Product usage data [21], [23], [33], [34] such as telemetry or product logs enable software companies to be more accurate in evaluating their ideas, and to move away from assumptions and towards trustworthy data-driven decision making [29], [34], [55].

The internet connectivity of software products provides an unprecedented opportunity to evaluate hypotheses in near real-time, and make causal conclusions between the changes made to the product and the customers' reactions on them [14], [18], [130], [136]. The way in which causal conclusions can be made in connected products is through A/B testing, or more generally, Online Controlled Experiments (OCEs) [29], [130]. OCEs can transform

decision making into a scientific, evidence-driven process—rather than an intuitive reaction [11], [29]. And as many large online companies such as Google, Microsoft, Amazon report on the benefits that they experience due to the development of the experimentation practices in their companies [25], [47], [56]–[58], the reminder of the software industry struggles to learn how to evolve and grow their experimentation capabilities [4], [10], [36], [59].

Systematically conducting trustworthy experiments at large-scale requires a transformation that is far from intuitive and easy to achieve. Conducting many OCEs annually requires technical capabilities that need to be developed, organizational changes that ingrain experimentation with companies’ culture and practices, and alignments of business objectives with experiment metrics [9]. For example, conclusions can be entirely wrong when experiments interfere with each other, or when the randomization of users is biased by previous experiments [15], [16], [187]–[189].

Objective. The main objective of this chapter is to broaden the understanding of online experimentation and its evolution in the software industry. Specifically, we aim to (1) generalize our findings from prior research, and (2) provide practitioners with prescriptive guidelines that can help them in evaluating and evolving their experimentation capabilities. A capability in our context is the ability to effectively conduct large numbers of trustworthy experiments with informative outcomes.

The main research question that we strive to answer in this work is “how do large-scale software companies grow their experimentation capabilities?”

We answer our research question by building on our prior and ongoing research with Microsoft, and conducting a multi-case study with three other large-scale software companies that have evolved their experimentation capabilities from being able to conduct a few OCEs per year to tens, hundreds and thousands of them.

Contributions. The main contribution of this chapter, and the answer to our research question, is The Experimentation Growth Model. This model summarizes the prerequisites for conducting

OCEs, describes the seven most critical dimensions of experimentation, prescribes the actions that practitioners can do to further advance their experimentation capabilities, and discusses a few concerns in conducting OCEs. This model extends our initial work that was induced from the single-case study at Microsoft with a new dimension of evolution (the impact of experimentation) and redefines existing dimensions of the model based on the feedback from several companies that applied our work to their context. Online software companies can use our model to assess their maturity for experimentation, and to learn how to grow the experimentation practices in their companies.

8.2 Background

8.2.1 Learning what customers Value

The topic of customer feedback (in this chapter, and for the purpose of our work, we use the terms ‘customer’ and ‘user’ interchangeably, however, we acknowledge that in some domains the two terms can have different meanings) has been increasingly studied in Requirements Engineering [62]–[66], Information Systems [67], [68], Human-Computer Interaction [69], [70], Product Management [40], [71], Entrepreneurship [31], [72], and Big Data Analytics [29], [190]. The term *feedback* is widely used in Requirements Engineering research to refer to the comments and evaluations that users express upon experiencing the use of a product or service [64], [65]. In the Data Analytics research, the term frequently appears in the context of measuring customer satisfaction [14], [183].

In previous work, researchers differentiate between *expressed feedback* (data that was either written, pronounced, or gestured in some way by the customers) [52], [62]–[66], [90], [182], [191] and *measured feedback* (data obtained by observing or measuring customer actions) [9], [15], [57], [58], [154], [192], [193]. In the case of former, the feedback data is typically qualitative in nature and is rich with opinions, wishes, and ideas that customers have about their problem and the solution. In the latter, the data is typically

quantitative and consists of signals (aggregated clicks, dwell times, etc.) or metrics that compute these signals over some given units (for example, clicks per user, transactions completed) [44], [122], [183].

Based on our empirical findings [9], [36], [156] and learnings from other researchers [10], [104], [113], [179], both types of feedback have their profound impact in the software development process and are important for successful software product development. However, and although both types of feedback complement each other, there is a fundamental difference in their accuracy, and the types of conclusions that we can make when using either of them. While the expressed feedback focuses on what the customers *say they do*, the measured feedback reveals what they *actually do* with the product [10], [40], [183]. A study on software privacy policies conducted in 2016 [194] further points out that there is a gap between what people say they wish and what they actually want. In this study, on a sample of (N=543) individuals, 98% of participants stated that they agree on donating their firstborn child in exchange of an access to a social network. Again, what customers say can differ from their actual intent, and using measured feedback data (e.g. product logs) when making decisions is recommended on top of the expressed feedback [195]–[197].

8.2.2 From Data to Valuable Insights

Advancements in Continuous Integration [7], [49], [50] and Continuous Deployment [51]–[54] enable companies to quickly build enhanced versions and variants of their products, and deploy them to their customers over the Internet. The ability to communicate with the product in the field, however, provides an unprecedented opportunity that enables software companies not only to deploy, but also to collect data (for example, product logs). Product log data can reveal the actual state of the product and its usage patterns [21], [23]. In the past, when software companies reasoned about “log data,” they often referred to the output from trace statements or crashes in source code, which help with troubleshooting and debugging [22], [195], [198]. Today, the log data are typically com-

plemented with telemetry, which normally consists of timestamped, customizable records that capture system and user events [23]. This setup enables software companies not only to identify the bugs and crashes in a particular version of a product, but also to reconstruct how their customers use the product [23], [57].

Companies can benefit from the data that are being collected from the products in several different ways. One possibility is through exploratory data analysis, which helps companies understand the data when little or no hypotheses exist, or when specific hypotheses exist but supplemental representations are needed to ensure the interpretability of the results [114]. Explorative data analysis can help in identifying, among others, customer demographics, how often they use the product, what version of the product is most popular in a certain segment, and finding other useful insights [45], [114], [115]. To discover useful insights through exploratory data analysis (and also other ways of reasoning around data that we discuss further in this chapter), analysts can create diagrams, charts, histograms, or other types of visualizations. Overall, the goal of exploratory data analysis in product development is to identify insights that may lead to hypotheses about the products and customers using them [40], [116]. To understand the customer behavior and discover relationships between multiple variables (for example, the version of a product and a customer segment), analysts can organize the data at hand in pairs (or even higher dimensions), and evaluate whether the given variables are correlated. To evaluate relationships, analysts can use common measures such as Person correlation coefficient, multiple regression, or other statistical techniques [117]. An important limitation of correlational designs, however, is that they cannot be used to draw conclusions about the causal relationships among the measured variables [118]. To make accurate decisions in product management, and conclude that a change to a product (for example, a new feature) is the cause that customers reacted as they did (for example, use the product more frequently), companies need to employ controlled experimentation [130].

8.2.3 Online Controlled Experimentation

“Our success at Amazon is a function of how many experiments we do per year, per month, per week, per day...”

-- Jeff Bezos, CEO @ Amazon [1]

As highlighted in the previous section, classical knowledge discovery and data mining techniques for data exploration can provide useful insight. However, the patterns discovered are correlational and therefore pose challenges in separating useful actionable patterns from those caused by “leaks” [154]. To establish a causal relationship between the changes made in the different variants of the product and the measures of interest (customer reactions), companies can conduct controlled experiments.

A controlled experiment is a scientific method that has been used for decades in the pharma industry to test the effectiveness of new drugs [16], [130]. In a typical controlled experiment, two groups are created by randomly assigning participants in either of two conditions: the control and the treatment. The differences between the two conditions are formalized as independent variables. For example, if the two variants are software products, they might have different design solutions, communicate with a different server, or have another independent variable that has been changed. In a standard, online controlled experiment (also known as A/B test or online randomized trial), users of a software product are randomly assigned between several variants of a product in a consistent and persistent manner (a user receives the same experience multiple times). User interactions with the product (e.g. clicks, dwell times, etc.) are instrumented and key metrics (e.g. engagement, task success, etc.) are computed [183]. The differences between the experiment variants (for example, the control and the treatment) are evaluated on a metrics level using statistical tests (for example, using a t-test).

The aim of this setup is to identify which of the variants, if any, yields improvements to metrics that the product and portfolio team strive to impact [29], [56], [199]. Since the assignment to experiment groups is random, the only things that could explain a differ-

ence between them is either a causal effect or random chance. If we can effectively rule out chance, we can argue that we have evidence for causality. In this scenario, statistical significance serves to rule out chance. And although designing a few isolated experiments to compute statistical significance may be easy for companies with statisticians [18], [200], systematically conducting trustworthy experiments at large-scale requires a transformation that is far from intuitive and easy to achieve [29], [189].

As we learned in [9], conducting hundreds or thousands of OCEs annually requires technical capabilities that need to be developed, organizational changes that need to be done, and business alignment that needs to be achieved to experience the most benefits from experimentation. Our initial work [9] can help companies in evolving their experimentation practices. However, it is based on the experience of product teams from a single case company – Microsoft, and as we learned through feedback that we received from other software companies, it omits certain details and dimensions that characterize the growth of experimentation capability.

In this chapter, and based on an in-depth case study at Microsoft and three other software companies, we generalize our findings from prior research. Specifically, we provide detailed prescriptive guidelines that help to evolve experimentation capabilities. Our main contribution is The Experiment Growth Model, which addresses the seven most critical aspects of experimentation. We start by first presenting our research approach in the next section.

8.3 Research Methodology

The findings reported in this chapter build on case study research [137], [139] in close collaboration with four large-scale, multinational, online software development companies. The study was initiated in April 2016, and the period we report on in this chapter covers April 2016 to October 2017.

8.3.1 Research Process

Our research process consisted of two major phases. In the first phase, we conducted an in-depth single case study at Microsoft. The outcome of the first phase was the Experimentation Evolution Model [9]. The second phase of this research consisted of a follow-up multiple case study, where we worked with Microsoft and with representatives from three additional large-scale online software companies to further detail and generalize our initial findings from phase 1. The result of this phase 2 is a redefined and extended Experimentation Growth Model, which we present in detail in Section 4.

Figure 9. Research process.

In the remainder of this section, we describe each of the two phases in greater detail. We introduce the objective and the context of each phase, present our data collection and analysis practices, and highlight the main outcomes.

8.3.2 Phase 1 – In-Depth Single-Case Study at Microsoft

Objective & Context. The aim of phase 1 was to understand how product teams at Microsoft evolved their experimentation practices. To achieve this, we conducted an in-depth inductive case study with the Analysis and Experimentation (A&E) team at Microsoft (from hereafter labelled as case company H). The A&E team provides a platform and service for running controlled experiments for their customers (the product teams). Its data scientists, engineers and program managers are involved with partner teams and departments across Microsoft on a daily basis. The participants involved in this research phase were primarily collaborating with the following Microsoft product and services teams: Bing, Cortana, Office, MSN.com, Skype and Xbox.

Protocol and Data Collection. The results of Phase 1 are based on historical data points that were collected over a period of two years at case company A, and complemented with a series of semi-structured interviews, observations, and participatory meeting. The first author of this chapter worked with the Microsoft Analysis & Experimentation team for a period of 10 weeks. During that time, he collected *documents, presentations, meeting minutes and other notes* available to Microsoft employees about the past controlled experiments, the development of the experimentation platform, and of the organization, conducted at Microsoft A&E over the last 5 years. In cumulative, we collected approximately 130 pages of qualitative data (including figures and illustrations).

The second author of this chapter has been working with the Analysis & Experimentation team at Microsoft for seven years. He was the main contact person for the other researchers throughout the data collection and analysis period and advised the diverse selection of data scientist, managers and software engineers that we interviewed. In total, we conducted 14 semi-structured interviews (1 woman, 13 men) using a questionnaire guide with 11 open-ended questions. The participants that work with different product teams were invited for a half an hour interview by the first two authors. The interview format started with an introduction and a

short explanation of the research being conducted. Participants were then asked on their experience with conducting controlled experiments, how they document learnings from those experiments, and how their practices changed over time. We also asked for examples of successes, pitfalls, and pain points that they experience while conducting controlled experiments.

Data Analysis. In this phase, we analyzed our data in two steps, following the thematic coding approach [180], [201]. First, we grouped the data that belonged to a certain product. Meeting minutes and interview transcriptions were independently assessed by three researchers. Next, we grouped products in 4 buckets based on the number of experiments that their product teams are capable of executing per annum (i.e. 1-9, 10-99, 100-999, and 1000+). Second, and with the goal to model the evolution of continuous experimentation, we performed inductive category development [161]. In the first step, six definitions of categories emerged, that represent our research interest. Next, we formalized the subcategories under each of the categories by reading through the collected data and assigning codes to concepts that appeared in it. Finally, we used the data collected to develop individual categories in greater detail.

Outcome. The main outcome of phase 1 are the initial guidelines for the evolution of controlled experimentation [9].

8.3.3 Phase 2 – Multiple-Case Study

Objective & Context. The main objective of phase 2 was to expand the knowledge of experimentation evolution and generalize our results from phase 1. In December 2016, we shared our research results from phase 1 with other software companies that conduct OCEs and solicited their interest in providing feedback to our research. Three software companies expressed interest to collaborate with us in order to further detail and generalize our work findings from phase 1.

Data Collection. In this phase, we collected the empirical data through *workshops*, *meetings*, *emails*, *Skype*, and direct messages with the individual participants. In total, we collected 235 pages of

qualitative data. The combination of the multiple data collection methods enabled us to explore the most critical aspects of experimentation in width (e.g. at workshops & meetings), and in-depth (e.g. through emails and direct messages).

Case Company H: The first author of this chapter collaborated with the second author for another three months in a similar engagement that we described in Phase 1 above with the case Company A - Microsoft. We collected additional data by taking notes at internal meetings, team off-sites, by examining recent experiments, conducting 22 additional interviews, and by engaging and solving challenges in experimentation.

Case Company I: With Booking.com, which is a large-scale online accommodation provider based in Amsterdam, Europe, we held a full day workshop, communicated with them using Skype, and exchanged several emails and messages through LinkedIn. Specifically, we met and engaged with a Senior Product Owner for experimentation, Chief Technology Officer, and the Director of Product Technology. Booking.com is the world's largest online travel agent and millions of users visit its e-shop to reserve their accommodation.

Case Company J: Our third company is Skyscanner, which is a large-scale online travel-fare provider based in Europe. With representatives from this case company, we held a full-day workshop, and communicated with them through Skype, email and LinkedIn messages. In particular, we met and collaborated with the Director of Experimentation, Senior Data Scientist Manager, Data Scientist, and two software engineers. All of our meeting notes were shared on Google drive and examined, commented on, and complemented by the contacts that work at Company C.

Case Company K: Our final case company is Intuit, which is a large-scale business and financial software company based in the US. With Intuit, we held a full-day workshop, and communicated with them through LinkedIn messages. In particular, we met with the Director of Analytics and Experimentation.

The workshops at the three additional case companies were similar in setup. We started by presenting our findings from phase 1 of this research. Next, we (2) asked the participants about the maturity and the growth process of the experimentation practices in their companies. Finally, we (2) asked for feedback on our initial findings. Specifically, we were interested in discovering what requires a more in-depth explanation, what we may have missed, and what needs to be changed.

Data Analysis. During the workshop, at least two authors of this chapter have been present, took notes, and have jointly analyzed the data using qualitative thematic synthesis [201]. Codes from phase 1 were used as the baseline, and new codes were created when the data that could not be semantically grouped under the existing codifications. In particular, one dimension of experimentation evolution that we did not create earlier was inductively derived in this way, and a few other updates were made to the initial experimentation evolution guidelines from phase 1.

Company representatives from the case companies constantly provided feedback on our analysis by clarifying any ambiguous term in the empirical data and confirming the most critical concerns identified in our analysis. This is known as the ‘*venting method*’ [77] and is valuable in situations where multiple experts can judge the same evidence and provide clarifications on it.

Outcome. By collaborating with the additional three case companies, we evaluated our initial recommendations on how to evolve experimentation practices in software companies. Along the way, we learned that our initial work omits an important dimension of experimentation evolution (the impact of OCEs), and that companies require more granular descriptions of the individual steps of the evolution. The main outcome of this research work, which spans across the two phases is the Experimentation Growth Model.

8.3.4 Discussion on the Validity

8.3.4.1 *Construct Validity*

We did a number of steps to improve the construct validity of our research. First, we conducted this research with four case companies that evolved their experimentation capabilities from being able to conduct a few online controlled experiments yearly towards being able to conduct hundreds or thousands of OCEs. Second, we worked with representatives in different roles (in particular, with engineering, project management, data scientists, line management, product management and experimentation directors). All of the participants of this study are very familiar with OCEs. In fact, the second, third and fourth authors of this chapter are permanently employed at the case companies. This setup enabled continuous access, insight, evaluations and confirmation of our analysis. However, and since this approach differs from a traditional case study [137], the contribution of our work risks in being biased as drawing extensively from an inside view. Despite this concern, this setup enabled us to detail our main contribution to the extent and accuracy that would not be possible otherwise.

8.3.4.2 *Internal Validity*

The threat to internal validity associates itself with the causal effect of the independent variables on the dependent variable. This effect can only be evaluated with the aid of a controlled experiment. Unfortunately, comparing the growth of experimentation practices through a controlled experiment would be very resource demanding in practice. One way to evaluate this would be to monitor companies that used our model and companies that did not use our model in their experimentation evolution and report on the difference between the two groups. Since this was not the primary aim of our study, we did not evaluate the causal effect of our contributions and findings.

8.3.4.3 *External Validity*

The external validity is concerned with the extent to which the results of the study are generalizable. By conducting a multi-case study, we improved the generalizability of our work. Specifically, we believe that the main result of this chapter – The Experimentation Growth Model – is applicable to online software companies with many users. Our learnings are transferable also to other domains, however, we cannot guarantee that our model captures all of the critical aspects in other contexts.

8.4 **Empirical Data**

In this section, we provide an insight into our empirical data. As described in the previous section, we analyzed hundreds of Online Controlled Experiments, word documents, emails, messages and meeting notes from the four case companies that are a part of this research. In our prior research [9], and to illustrate the evolution from the least to the most mature experimentation capabilities, we organized the empirical data following the capabilities of each of the product teams at Company A. In this chapter, and to further stress the insights that we learned by working with the case companies, we organize the empirical data in learnings. This section is not exhaustive, however, it provides an insight into the empirical

foundations on which our Experimentation Growth Model is build.

We illustrate each of the learnings with example experiments, quotes, and other insights from our empirical database. Although we could present many learnings and experiments, we (1) selected five learnings that are most frequently mentioned in our empirical data, and we (2) use illustrative experiments that our case companies are comfortable with sharing publicly.

8.4.1 L1: Data Quality Fosters Trustworthiness

One of the main reasons on why product teams in our case companies conduct OCEs is their scientific trustworthiness. Trustworthiness, however, is fostered with the data that need to be systematically collected and correctly analyzed. Data quality issues are common in experimentation, and detecting them during experiment analysis is critical from the start, and growing in importance as companies scale their experimentation capabilities.

“You have to constantly prove that the tool is accurate. It is easy to blame the data. When scaling, the accuracy of the experimentation platform has to go much up.”

--Case Company I

We illustrate the importance of this learning with two example experiments.

Office Contextual Bar Experiment. Microsoft Office is a well-known suite of products designed for increasing work productivity. Data-driven practices in Office product teams were at the time of this experiment in the early stages. The product team responsible for the edit interface in Office mobile apps conducted a design experiment on their Word, Excel, and PowerPoint apps. The teams believed that introducing a Contextual Command Bar (see Figure 10 below) would increase the engagement compared to a version of the product without the contextual bar. Their hypothesis was that mobile phone users will do more editing on the phone because the contextual command bar will improve editing efficiency, which will result in increased commonality and frequency of edits and 2-week retention.

Figure 10. Office mobile contextual bar experiment.

During the set-up of the experiment, the team ran into issues with measuring the number of edits. The instrumentation was still in the early stages, and the telemetry teams did not accurately log the edit events. These issues had to be fixed prior to the start of the experiment. The results of a two-week experiment that could be conducted after fixing the data-collection issues indicated a substantial increase in engagement (counts of edits), but no statistically significant change in 2-week retention. The experiment provided the

team with a key learning that resonates across the product teams at Company A, and the learning frequently emphasized by the other case companies: **proper instrumentation of features is essential for high quality data collection, which is needed for the computing experiment metrics.**

MSN Carousel Experiment. In another experiment at MSN, the product team increased the number of its rotating cards on the MSN Carousel located on the MSN homepage (specifically, the increase was from 12 to 16 cards). Contrary to expectations, the experiment results were initially very negative. This suggested that the 16 cards carousel is worse than the carousel with 12 cards. After performing a data-quality check that evaluates whether the observed number of users in the treatment and the control group equaled the expected ratio (SRM check), the team learned that users in the treatment group were engaging with the cards to the extent that (1) they have been classified as bots, and they were (2) consequently excluded from the analysis.

“The results were essentially flipped after accounting for the bot classification.”

--Case Company H

The experiment provided a learning that is frequently emphasized by all of the case companies: every analysis requires data-quality checks to pass before the results of the experiment can be trusted.

8.4.2 L2: Platform Accelerates Experimentation Growth

All of the four case companies are constantly developing and improving their experimentation platforms. An experimentation platform enables to quickly start, iterate, and analyze experiments, and at the same time abstracts detailed experimental concepts from the experimenters. By improving the experimentation platform and its functionalities, the case companies could grow the number of experiments that can be conducted in parallel and at the same time, improve their quality.

“The challenge first is to do experimentation. Next, the challenge is to do it easy.”

--Case Company I

Although there are many features that an experimentation platform should have as companies grow their experimentation capabilities (which we present in the next section in greater detail), we highlight one below.

Learning from past experiments: One of the features that all of our case companies frequently emphasize as important at their stage of experimentation growth is the ability to accumulate the learnings from their experiments through the experimentation platform. This challenge is not novel neither specific to experimentation. However, by enabling everyone to quickly learn what has been already tried, when, and how customers reacted on it, profoundly impacts the roadmap of future experiments.

“People must find past learnings a few years from now. That knowledge should not be lost!”

--Company I

“For any experiment, we can quickly see its scorecards, the change that was done, and the comments that the experimenters put in the system.”

--Company J

“There are two levels of knowledge here: how to do experiments, and what we learnt from doing experiments.”

--Company I

“We have a timeline feature that enables us to quickly see the order of experiments and their impact”

--Company J

8.4.3 L3: Qualitative Insights Enrich Experiment Learnings

Despite the accurate experimental methodology and millions of users that our case companies operate with every day, in reality many ideas yield no quantifiable impact to the key metrics. More precisely, they don't impact the metrics at hand. This leads into a situation where making a decision can be difficult:

“Sometimes one can get stuck in a data paralysis...We have all the experimental data and we do not know how to take a decision.”

--Case Company I

To address this limitation, our case companies consistently improve their metrics as they evolve their experimentation, and the potential to detect changes that they have not considered previously as relevant could be detected. On the mid-term, however, qualitative customer feedback can be helpful to sidestep the data paralysis. In fact, product teams in all of our case companies, regardless of their experimentation maturity, continuously collect qualitative feedback through surveys, emails, feedback options in their products, or in any other way that they can reach their users and ask them for their preferences.

The main use of these qualitative learnings in highly data-driven organizations is in interpreting ambiguous quantitative movements, improving metrics, and coming up with new hypotheses to test in an experiment. In the example below, we illustrate a controlled experiment from company C that benefited from such qualitative insight:

Direct Flight Experiment. In this experiment (see Figure 11), the product team evaluated a new feature that reveals their app users a possibility of a direct flight between two destinations. By analyzing the experiment results, Company I learned that users of the application do not find the new feature more helpful (for example, they do not change their initial date of travel to the days where direct flights exist) compared to the product without it. Even with large statistical power they could not detect an improvement to any of the key metrics for their product. Despite this neutral outcome, however, Company C decided to deploy the feature to everyone. The reason why this has happened is because they received large amounts of positive qualitative feedback from their customers that were exposed to the treatment, which gave the product management the confidence that this is the right decision.

Figure 11. Direct flight experiment from case company J.

“Feature looks [based on experiment results] like a wash. However, people have written to us about how great this is. Skyscanner ships.”

-- Company J

8.4.4 L4: Iterating OCEs with Small Changes Yields Large ROI

Product teams at our case companies frequently invest resources into developing new features for their products. What they learned through experimentation, however, is that small improvements to features, which are inexpensive to implement compared to the initial development of a new feature, can have a tremendous ROI. In some cases, the absolute impact of small improvements can exceed the impact of the initial feature introduction. Our case companies therefore frequently iterate their experiments to explore the impact of small and inexpensive changes in various segments. We illustrate this learning through an example experiment from case company I.

Email Campaign Experiment. In an experiment conducted at case company I, the impact of an email campaign was evaluated [202]. Email campaigns are a direct and economical way to expose customers to online products. In the case of case company I, these emails contained propositions of travel destinations and information about special deals from partner hotels. Both personalized recommendations and deals have an obvious value to the user, however, the repeating nature of the emails makes the campaigns effectiveness wear-off over time. This particular experiment consisted of two iterations. In the first iteration, 34 million users were exposed to two variations of the email. Variation A contained the most popular endorsements, whereas variation B was a personalized email that used a Machine Learning algorithm to identify the best recommendations for the receiving customer based on their past trips.

Figure 12. Personalized email experiment.

The first iteration was very beneficial in multiple metrics, including users clicking on the campaign, interacting with the website and finally converting from visitors to bookers. More specifically, case company B observed an 18% uplift in clicks at the same bounce rate, 10% uplift in net conversion, and 14% increase in cancellations (22% increase in gross conversion).

The bold statement in variation B, however, caused vivid responses among the people who were exposed to the experiment. Specifically, the statement *“our team of travel scientists ... based on your past adventures they think that you have a passion for..”* caused, among others, skepticism about the data that case company B access to (see Figure 13). Some customers also expressed concerns around the idea that human beings were involved (even though an algorithm made the recommendations, the copy suggested “travel scientists” were involved) in reviewing their past trips and selecting the recommendations.

Figure 13. Personalized email experiment – Customer Responses on Twitter.

The qualitative insights were, as with many experiments, used to drive another iteration of the experiment. In a second iteration, customers were exposed to two variations that only differed in the way the copy that accompanied the personalized recommendations was phrased. The first variation was identical to variation B from the first iteration. Variation B in the second iteration of this experiment, however, contained a less intrusive message promoting endorsements in suggested destinations without a direct reference to how these endorsements were selected. Specifically, any reference to "travel scientists", or other humans involved in generating the recommendations, was removed. This minor adjustment caused an impressive additional increase in net and gross conversion (+7.14% and 10.5%) and a modest increase in cancellations (+4.39%), nearly doubling the impact of the original implementation.

“People think that we must have changed something about the model, made the recommendations smarter, something like that, but we didn’t, the recommendations themselves were identical. Copy and framing have tremendous power, but you will not learn that unless you start measuring their effects.”

-- Case company B

To execute the first iteration of the experiment, the product teams had to (1) build a machine learning model and (2) design an email campaign. Without any doubt, this was a significant engineering effort that resulted in an impressive impact. On the other hand, the second iteration of the experiment only improved the wording in an email. As a result, however, it nearly doubled the impact of the experiment. In fact, more and more recommender systems get evaluated not only on their accuracy, but also in their contribution to the user-satisfaction metrics. In this scenario, explaining recommendations to users can be as important as the recommendation itself [203].

The example experiment above **illustrates the potential of small ideas and improvements**. Just as importantly, it demonstrates the benefit of having a capability for running many experiments cheaply, concurrently, and in iterations where qualitative insights accumulate and complement experiment learnings.

8.4.5 L5: The Impact of Experimentation is Boundless

Many experiments that are shared on conferences and in research papers typically showcase a few design variations that led to better conversion (e.g., online store checkout). Although this is one area where experimentation yields many benefits, mature case companies benefit from experimentation in many other ways. OCEs in our case companies are also commonly conducted to determine relevance (e.g. in sorting search results on a page), projecting infrastructure capacity (e.g. projecting the impact of the change), improving the quality of their products, and planning team efforts

(e.g. deciding which features to add/remove/improve next). We present a few common examples of experimentation impact in greater detail next.

Projecting Infrastructure Capacity: We illustrate this benefit with an OCE conducted with Skype at case company A. Skype is a well-known product for video and audio communication. Recently, a feature team completed migration of the product to a new call-routing algorithm. Among other effects, the team hypothesized that enough infrastructure capacity will be available to support the new approach to call routing. To evaluate their hypothesis, the Skype team conducted an OCE with only a few percent of Skype users. The difference in load on the infrastructure between the new and the old algorithm was estimated, and projected to 100% of user population, confirming that the capacity will be sufficient. With this, the Skype team learned that based on the effect of the limited treatment group, enough capacity is available within the new infrastructure to continue with the transition. By identifying the exact effect on the infrastructure, the team was confident that the infrastructure capacity is within the range needed for the complete product migration.

Improving Product Quality: Contrary to intuition, conducting OCEs and acting on them accordingly can only increase the quality of the products that are being developed. As companies release new variations of the product to real-life customers, statistical significance between them can reveal surprising bugs and issues that are very difficult to discover in a pre-production environment. We illustrate this with an example experiment from case company A.

Figure 14. MSN Video Player OCE.

MSN team at case company H created a version of the video player that covers a larger area of the page than its control variation (see Figure 14). The team hypothesized that the larger video player will enable a more distraction-free experience. An OCE was designed with one treatment and one control and iterated two times. The first iteration of the experiment caused an alarm indicating a difference between the observed and the expected split of users between the variants. The analysis of the experiment data revealed a bug with the product's configuration files. The MSN team learned through an experiment that a previously unknown bug exists in their product. By removing the bug, they learned something that all of our case companies recognize: **experimentation increases the quality of the product under an OCE.**

Planning Team Efforts: Mature experimentation practices also profoundly change how feature team effort is planned and measured. Instead of ordering feature teams exactly what to develop, our case companies provide individual feature teams with metrics that they should target. As OCEs reveal the exact amount of contributions to those metrics, feature team success can be accurately measured and rewarded. One example of this is Bing's relevance team at case company H, whose yearly goal is set in terms of specific improvement to the metric measuring users' search success. Each sub-team, e.g. core result relevance, related searches, query suggestions, etc. gets assigned a part of the goal, which is measured as the sum total of improvements to the metric over all the experiments the team ran and deployed.

On top of sharing the outcomes of individual OCEs within and between feature teams, our case companies evangelize their impact across the company and, in one of the case companies, even with the board of directors.

"...If you recall the game plan that we shared at our Investor Day in the fall, we outlined the plan to extend our leadership in do-it-yourself tax, began transforming the assisted tax category and **ran experiments to evolve beyond tax through a tested open platform**. This season, we made progress across several fronts...."

-- Case Company K [204]

In the next section, we generalize our work based on all of the empirical data that we have collected and analyzed from our four case companies. As a result, we present a stepwise model to grow experimentation practices in online software companies.

8.5 The Experimentation Growth Model

In this section, and to answer our research question - “how do large-scale software companies grow their experimentation capabilities?”, we present our main contribution: The Experimentation Growth Model. With this label, we highlight the fact that experimentation practice evolves and grows along multiple dimensions within the software company. As indicated by the learnings in the previous section, companies that strive to conduct OCEs at scale should continuously develop and improve their experimentation platform, record experiment learnings, iterate experiments, and expand the impact of their OCEs. The Experimentation Growth Model summarizes these learnings in a progressive manner. The model consists of two parts; the prerequisites for experimentation, and the four stages of experimentation evolution. We discuss each of the two parts next, and follow up with a recommendation on how to use our model as guidelines in software companies.

8.5.1 Prerequisites for Experimentation

Based on our empirical data, we identified three prerequisites for conducting OCEs. To conduct trustworthy experimentation, companies need to (1) understand the statistical foundations of controlled experimentation, (2) ensure connectivity of to their products while they are in the hands of their users, and (3) have the psychological safety that will enable them to celebrate the actual success and learn from the experiments that do not yield the desired outcome.

Statistical Foundations. Experimentation is a scientific, evidence-driven process that requires a profound understanding of statistical fundamentals of hypothesis testing. The common role in 2017 with statistical skills for experimentation is Data Scientist [18], [133]. In fact, skills for experimental design have been mentioned as one of the most critical demands for Data Scientists in a recent survey [18]. And since this position is expected to be the most difficult to fill in for software companies in the next 5-year period [134], we highlight the essential statistical foundations for online controlled

experimental design. Specifically, *Power Calculation*, *Chi-square test*, and *t-test*.

With the *Power Calculation*, experimenters identify the minimal number of experimental units (for example, users) that they need in the experiment groups in order to detect the desired difference (also known as an effect size). For example, to detect a 1% difference between two groups, far more randomization units are needed in the experiment compared to if, say 10% difference is the objective. Power calculation is in detailed described in [205] and an open-source calculator is available at [206].

Second, the chi-square test is used in experimentation to detect critical data quality issues, specifically, to perform the **Sample Ratio Mismatch (SRM) Calculation**. In this calculation, expected distributions between the experimental groups (for example, 50% for the control group and 50% for the treatment group) are compared with the actual observed distributions (for example, 51%/49%) using the chi-square test. If the SRM calculation reveals a statistically significant difference (for example, the chi-square test returns a p-value lower than conventional 0.05), experiment results should not be trusted as the number of units in each of the groups differs from the expected.

Finally, and to compute the **Statistical Difference** between two groups with independently and identically distributed units, a student's t-test can be used. A student t-test in the OCE scenario takes a vector of values for each group (for example, a vector with a count of clicks for the control and the treatment groups) and compares the means within the two groups, returning a p-value - the probability to observe a difference between those means equal to or larger than the one actually observed, under the assumption that there should be none.

In both of the tests above, conventionally, a p-value lower than 0.05 suggests statistical significance (the assumption that there is no difference can be rejected), and a p-value lower than 0.001 suggests high statistical significance. Both chi-square and t-test are common in statistical packages and their implementations are available in most of the programming languages. As reported in re-

cent research [189], companies risk experiencing many pitfalls of experimentation should they rely solely on use of the platform without a fundamental understanding of the underlying statistical concepts presented above.

Connectivity of Products. To conduct OCEs with software products, companies should be able to establish, at least periodically, a bidirectional connection between the software product and the backend services in their organization. Some products are deployed as web services in the cloud infrastructure by nature (e.g. Skyscanner.com search engine) and satisfy this requirement. Other products such as client applications (for example Microsoft Word) may traditionally not contain the capability to query the backend infrastructure for software changes or send contextual logs to the server. In the next section, we explain in detail what type of data exchange is required in a certain state of experimentation evolution. As a prerequisite, however, companies that wish to conduct OCEs should design their products in a way that enables them to (1) identify their product instances remotely (for example, by giving each instance a unique identifier), be (2) able to update the product remotely (for example, to deploy an incremental update), and finally (3) product instances should be able to send back the telemetry data.

Psychological Safety. To enable a full integration of experimentation into the software development cycle, companies should exercise and build psychological safety. Psychological safety is a shared belief held by team members that the team is safe (for example, for interpersonal risk-taking and experimenting) [207]. For decades, companies have considered a deployment of a new version of a software product or a new software feature as a success. However, when experimentation is introduced, declaring success is reserved for features that deliver measurable customer value. As many ideas will be disproven when experimentation is introduced, it is critical for engineers, designers, managers and project leaders to be prepared in learning from those experiments, to accept that most of the ideas fail to do what they were intended to do, and to

be nimble in what to try next. Most importantly, the experimenters need to be comfortable in sharing the work that they have done. Especially the work that does not have the desired impact on the product, or the work that harmed the product.

To provide an illustrative example, we reference a recent study [208] that used controlled experimentation to disprove a decade known “truth”. For many years, the medical community believed that operating hearth patients using percutaneous coronary intervention (PCI) in a stable angina provides the many benefits of better quality of health. In this recent study, however, a randomized experiment indicated that there is no difference between the group that undergone the procedure (a stent was inserted in their artery) and the placebo group that did not. Should the doctors continue to suggest this risky operation to their surgeon colleagues and their patients? Luckily for most of the online software products, human lives are not at stake. However, and to disprove “truths” that exist within software organizations and learn from this, strong psychological safety is required. In fact, strong psychological safety has proven to be the key factor that empowers teams to challenge known facts and collectively learn faster many years ago [207]. We observe in our case companies that the same holds today, especially with experimentation. Fortunately, there are multiple steps that companies can do to accelerate the growth of psychological safety such as actively replacing blame with curiosity, asking for feedback on delivery of product features, and in other ways described in [209].

8.5.2 Experimentation Growth

In the previous section, we described the prerequisites for experimentation. In this section, we reveal how companies start and conduct their first OCE, and how they can grow their experimentation capabilities through the four stages of experimentation growth: ‘Crawl’, ‘Walk’, ‘Run’ and ‘Fly’. These labels emerged in phase 1 of this research to describe the four distinct stages of capability to experimentation:

‘Crawl’ is the initial stage where software companies start to experiment, which typically happens within a single feature team and without an experimentation platform.

In the ‘Walk’ stage, companies develop the capability to conduct tens of experiments per annum.

In the ‘Run’ stage, multiple feature teams conduct hundreds of experiments yearly with an experimentation platform that controls for common pitfalls and alerts experimenters for bad experiments.

Finally, in the ‘Fly’ stage, software companies are able to conduct thousands of OCEs yearly, evaluating every bug fix through a controlled experiment for unexpected outcomes.

Below we will discuss these four stages of experimentation by elaborating on them according to the seven most critical dimensions of experimentation identified in our data analysis process through inductive category development as described in the method section. Those seven dimensions can be summarized as follows:

- DM1: **‘Technical focus of product development activities’**: consists in the engineering work that enables experimentation growth in a certain stage (for example, building a reliable data pipeline).
- DM2: **‘Experimentation platform capability’**: focused on the features that companies need in their experimentation platform.
- DM3: **‘Experimentation pervasiveness’** assesses the extent to which experimentation is being conducted in software companies (for example, from a few user interface experiments towards experimenting with every change).
- DM4: **‘Feature team self-sufficiency’** assesses the extent to which individual feature teams manage the experiments without the Data Scientist involvement (for example, growing from creating- towards analyzing experiments).
- DM5: **‘Experimentation team organization’** describes how to organize the experimentation experts in each of the four stages, in order to experience the most benefit from conducting OCEs (for example, from no experimentation team to a center of excellence).
- DM6: **The ‘Overall Evaluation Criteria (OEC)’** dimension entails how the key metrics are defined (for example, from a few signals to a structured and well-defined set of metrics).
- DM7: **‘Experimentation Impact’** reveals the extent to which the experimentation impacts the software company at each of the four stages (for example, from an impact to a small feature, towards an impact that sets strategic and team goals).

Those seven most critical dimensions of experimentation are used as rows in Figure 15, and we discuss in greater detail next how each unfolds on the four stages of growth.

	Crawl	Walk	Run	Fly
Technical focus	Data Collection: From <i>debug logging</i> to <i>contextual logging</i> .	Standardization. Creating a standard set of metrics: Success metrics, debug metrics, guardrail metrics, data quality metrics.	Comprehensiveness. Comprehensive data collection, reliable data pipeline, and an exhaustive set of metrics. Automated analysis of experiments.	Consolidation. Unified data pipeline enables easy creation of cross-team metrics. Standardized process for metric creation, evaluation and management.
Experimentation platform capability	None: No Experimentation platform needed.	Basic. 3rd party platform can be used or internally developed. Basic features such as Trustworthy Treatment Assignment, Power Analysis, Pre-Experiment A/A testing, are required.	Evolved. The experimentation platform should be extended with new features supporting larger volume of experiments, such as Alerting, Control of carry-over effects, Experiment iteration support, and metric creation. Teams manually record the most important learnings from experiments.	Advanced. Intelligent platform supporting features such as interaction control and detection, near real-time detection and automatic shutdown of harmful experiments, drill-down capabilities to understand metric movement and metric operations. Learnings from experiment are automatically recorded.
Experimentation pervasiveness	Visibility: A handful of experiments to showcase the value of experimentation	Scope. Broadening the types of experiments on a selected set of features (e.g. from design to performance, from performance to infrastructure).	Expanding. Experiments are conducted on the majority of new features developed for the product.	Complete. Experiment with all changes, even bug fixes.
Feature team self-sufficiency	Limited: Feature teams rely on Data Scientist to setup, manage and analyse experiments.	Experiment setup: Creating the experiment (instrumentation, A/A testing, assigning traffic) is managed by the feature team.	Experiment management: Feature team is responsible for monitoring experiments, making ramp-up and shut-down decisions, designing and deploying experiment-specific metrics.	Analysis. No Data Scientist involvement needed for most experiments. Experiment results are intuitive for feature team to interpret and make decisions.
Experimentation team organization	None. No experimentation team.	Standalone. Centralized experimentation team is created. No or very little experimentation skills in feature teams.	Embedded. Experimentation team data scientists are embedded into individual feature teams to become domain experts and grow experimentation in the team.	Partnership. Feature teams have their own experimentation experts that create a strong unity with business. Experimentation team serves as a center of excellence partnering with experimentation experts on feature teams.
Overall Evaluation Criteria (OEC)	Experiment Specific. OEC is defined for every experiment.	Defined. OEC is defined as a structured set of metrics consisting of Success, Guardrail and Data Quality metrics.	Evolved. OEC is validated over a large number of experiments and evolved with the findings from the learning experiments. Single metric as a weighted combination of others is desired.	Stable. Only periodic changes are allowed to an OEC (e.g. 1 per year). It is used to set the performance goals for teams within the organization.
Experimentation Impact	Feature Constrained to the team conducting incremental feature update.	Team. Starting to impact the team's activity planning (e.g. release cycle). The impact is shared within the larger team.	Multiple teams. Experimentation impacts infrastructure planning, product quality and product complexity (e.g. controlled feature removal).	Executive and External. The impact of experimentation is widespread across the product. Product teams are rewarded for experiment metric gains. Experiment results are shared with top executives, Board of Directors, and externally.

Figure 15. Experimentation Growth Model – The Evolution of Experimentation.

8.5.2.1 Crawl Stage

Technical focus of product development activities. When initiating experimentation growth, the first technical focus of product teams should be on implementing *contextual logging*. Most software products typically have logging in place for debugging or testing purposes [195]. The challenge with this type of logging, however, is that it does not necessarily reveal how the products are used. Hard to use data limits the number and types of metrics that experiment owners can define, and the number of ad-hoc analyses they will do, all resulting in sub-optimal decisions and fewer learnings obtained from the experiments. Therefore, logging procedures

should be updated by creating a centralized catalog of events in the form of class and enumeration, and implemented in the product code [23]. For having trustworthy analysis that can be reproduced overtime, the data should be stored in a standard and easy to consume form with a standard schema which can be relied upon.

The goal of such systematic logging is that a data scientist, analyst, or anyone else in the organization who is not familiar with the feature or the product itself, can understand what action or event was triggered by simply looking at the name of the event. Name of events should be consistent across products and platforms so that it is easy to search for them and link them with tangible actions in the product. Depending on the type of the product and the device that it is running on, different options for logging events may be available [21]. For products on the web or on mobile applications, events such as clicks, dwell periods (time spent on in a certain stage of a product), navigational actions (for examples mouse or finger position, scrolls or swipes) can be captured and recorded in a database of events. Event information should be aggregated and anonymized for its use in computing experiment metrics. This preserves privacy by enabling the deletion of raw events that could be associated with an individual user, or keeping the raw events only for a limited period. In the light of the data privacy and protection (for example, the EU GDPR policy) [210], the data collection system should also accommodate for users to view and manually delete the raw events on their request.

Experimentation platform capability. To start experimenting, an experimentation *platform* is not required. An initial experiment can be performed manually using open source tools for statistical analysis. In principal, three calculations should be done to analyze the initial experiments: *Power* calculation to determine the length of the experiment and the split of users, the *SRM* calculation to control for an unexpected difference between the experimental groups, and finally the *Statistical Difference* calculation to determine whether the experimental groups differ. Each of these calculations are described in the prerequisites above, and can be done us-

ing open source solutions that are available as statistical packages in most modern programming languages.

Experimentation pervasiveness. In this first stage, the value of experimentation is under scrutiny. Product teams that code their first experiment should focus in communicating the aim and the value of their learnings from the first experiments to their peers and managers. As an example, product teams can start with an experiment where multiple variations are already available and discussed among the team members.

Feature team self-sufficiency. When software organizations start to adopt the experimental methodology, the individual feature teams typically do not have the skills that are needed for trustworthy OCEs. Even if the teams early on decide for a commercial platform that tries to abstract these, they will not be able to understand the results of their experiments, nor if they should trust them [189]. One way to solve this challenge is to hire or educate a few individuals with the knowledge about the fundamental concepts of controlled experimentation (the ones presented in the Prerequisites section above). Individual feature teams will then rely on these individuals to setup, manage, and analyze experiments.

Experimentation team organization. In this first stage, companies do not have an experimentation team that could consult with individual feature teams. Instead, the individuals interested in experimentation may emerge within existing feature teams, or be hired to them as part of a cross functional team. These individuals should become very familiar with the prerequisites for experimentation, and start sharing their knowledge to others interested in OCEs.

Overall Evaluation Criteria (OEC). Software companies starting to experiment typically define their OEC for every experiment. These are not necessarily related. For example, a feature team experimenting with a User Interface might decide that increasing the number of clicks in one part of their application could be a good goal for their first experiment. In the next experiment, the OEC might be very different and unrelated to the one in the first experiment. The ideas for the initial experiments typically emerge organically within the feature team members, or have been expressed by the customers through interviews or other qualitative feedback collection techniques [211]. Experimenting with different OECs is strongly encouraged in this initial phase in order to learn how customers react to different types of changes, and to get an early indication on how much effort is needed to cause and detect a change in a certain metric. Prior research on pitfalls in metric interpretation provides valuable insights on choosing initial experimental metrics [16].

Experimentation Impact. In this early stage, contrary to all of the following stages, the impact of the initial experiments typically remains constrained to the feature that the OCE ran on. For example, a team working on improving a classification algorithm might impact the accuracy of their classifier, and a back-end team might improve the time to compute a search result. Since the OEC metrics in this stage are experiment specific and typically not aligned with the core business of the organization [15], the improved features are the main outcome of the first experiments, and their impact is very limited compared to what the companies can achieve by evolving to the next stage(s).

8.5.2.2 *Walk Stage*

Technical focus of product development activities. In contrast to the “Crawl” phase where individual experiments were evaluated by comparing the volume and distribution of signals such as clicks and page views, the focus in this phase should be on standardizing metrics in groups. Metrics are functions that take signals as an input and output a number per unit. Anonymized log signals should

first be categorized into classes and combined into metrics by being aggregated over analysis units (see e.g. Figure 16). Example classes are: **action signals** (e.g. clicks, views, visits), **time signals** (minutes per session, total time on site, load time), and **value signals** (revenue, units purchased, ads clicked).

Figure 16. Example of creating a metric for Experimentation.

The units of analysis vary depending on the context and product. The following apply at Microsoft for web products: per user (e.g. clicks per user), per session (e.g. minutes per session), per user-day (e.g. page views per day), and per view (e.g. clicks per page view). For other types of products, units of analysis might be different. For instance, for a well-known video-conferencing Microsoft product, a useful unit of analysis may be “per call”.

As the number of metrics grows, companies should organize their metrics in groups. Contextually the following four groups of metrics are useful for analyzing experiments: **Success Metrics** (the metrics that feature teams should improve), **Guardrail Metrics** (metrics that are constrained to a band and should not move outside of that band), **Data Quality Metrics** (the metrics that ensure that the experiments will be set-up correctly, and that no quality issues happened during an experiment), and **Debug Metrics** (the drill down into success and guardrail metrics). A research contribution from Google provides practical guidance on the creation of different types of metrics [183].

By grouping metrics together, deciding on experiment success can be done more accurately and effectively. For example, if data quality issues arise in an experiment, typically multiple metrics will respond and reflect on them. Examining these metrics together provides a quick insight into underlying conditions for the change.

Experimentation platform capability. In this stage, and as feature team start to create many metrics, experiments are becoming more informative. At the same time, examining many metrics together introduces a data quality concern known as multiple comparisons problem (aka multiple testing). Experimentation platform should inform the experimenters about this concern or, better, correct it. One way to correct for multiple testing is to apply the Bonferroni correction [212], however, other approaches that are not that restrictive are also possible.

Experimentation pervasiveness. In contrast to the “Crawl” phase where experiments are typically conducted between different design variants of a user interface or with the contents of the product, the main goal of this phase is to broaden the types of experiments on a selected set of features (e.g. from design to performance, from performance to infrastructure experiments). There are two main reasons why broadening the type of experiments in this stage is beneficial. First, by having feature teams experiment in different areas, the probability for experiments to interfere or interact with each other is low. Second,

Feature team self-sufficiency. In the “Walk” stage, feature teams should setup and start experiments on their own. The first step in this process should be to start A/A experiments (configuring the same experience for both control and the treatment groups) and observing the results of their experiments. In this way, individual feature teams don’t risk in introducing harm to their customers.

Experimentation team organization. A standalone, centralized experimentation team is created. No or very little experimentation skills in feature teams.

Overall Evaluation Criteria (OEC). OEC is defined as a structured set of metrics consisting of Success, Guardrail and Data

Quality metrics. Guidance for good OEC design and pitfalls of this process are well discussed and can be found in [14], [15].

Experimentation Impact. In contrast to the ‘Crawl’ stage where the impact of experimentation was limited to individual experiments, in this stage existing experiments start to impact the team's activity planning (e.g. release cycle) for future OCEs. Predicting impactful changes to the product is very difficult without conducting a few OCEs. By identifying the changes that showed the most impact on the product-wide metric, teams learn from each other and re-prioritize their work. The impact is shared within the larger team and future experiments are better planned in this way.

8.5.2.3 *Run Stage*

Technical focus of product development activities. In this stage, hundreds of OCEs are conducted, many running in parallel. Some of the experiments, however, may bias future experiments. For example, when users of a search application receive bad results, they bring their negative bias with them to the next experiment. This is known as a carry-over effect [15], [187]. To counter carry-over effects from previously completed experiments, newly conducted experiments should follow a template that first conducts an A/A test with the experimentation units. If this quality check reveals statistical significance in the key metrics, the experiment template might suggest to locally re-randomize the randomization units before the actual A/B test starts. The standard approach to ensure even distribution of carry-over effect is to make sure *every OCE is randomized independently*. This can be achieved by appending a unique hash salt for the randomization hash function for each experiment. When a subset of users for an experiment is selected, another independent randomization to split those users into treatment(s) and control may be needed.

Experimentation platform capability. In this stage, the experimentation platform should support *iterating experiments* (for example, with a higher percentage of assigned users), *control of carry-over effects* (detecting when users become biased due to a previ-

ous, for example, bad experiment), *easy creation of new metrics*, and *alerting*. Configuration of features variants as described in [213] might be helpful in accelerating the growth of experiments. When many experiments are conducted in parallel, however, having a manual overview of metric movements will become a resource-demanding task for experimenters. Automated alerting should be introduced together with the ability to divert traffic to control if an emergency situation occurs (e.g. a significant decrease of a success metric).

Experimentation pervasiveness. In contrast to the previous stages where experiments were conducted on demand with the reasoning to decide between a few different variants, in this stage the Experiments are conducted on the majority of new features developed for the product. Experimentation is seen as a vital step of the release cycle, approving feature deployments with positive effects, and preventing changes with negative or no effects on the OEC metrics to be deployed to the product. As seen in our empirical data, in situations where no OEC movement is detected, qualitative data can help feature teams to better understand the experiment outcome.

Feature team self-sufficiency. In contrast to the previous two stages, teams in the ‘Run’ stage should be responsible for setting up and managing their experiments. Managing experiments consist of starting the experiment, making ramp-up decisions (for example, know that it is safe to increase the treatment group size from 5% to 50%) using the experiment iteration feature, shutting the experiments down when they cause degradations, and deploying and editing metrics.

To enable feature teams to make better decisions in experiment management it is critical to provide them with intuitive scorecards. In a typical scorecard, the most relevant information about experiment metric movements should be displayed. If we take an experiment with one control and one treatment as an example, the scorecard should display, per every metric, the *result of the metric for the control and the treatment* (for example, an average of clicks), the *difference between the two* (sometimes expressed as a

percentage of change relative to the control - %delta), and most importantly, the *p-value*.

To make more informed decisions, useful breakdowns of metrics in scorecards are recommended. For example, and as visible in Figure 17 below, understanding why a success metric ‘Overall Clicks Rate’ has negatively reacted in the treatment group might be easier to interpret with the additional knowledge that it was caused by a decrease of clicks on ‘Web Results’, while other components of the page had click increases.

Metric	T	C	Delta (%)	<i>p</i> -value
Overall Click Rate	0.9206	0.9219	-0.14%	8e-11
Web Results	0.5743	0.5800	-0.98%	~0
Answers	0.1913	0.1901	+0.63%	5e-24
Image	0.0262	0.0261	+0.38%	0.1112
Video	0.0280	0.0278	+0.72%	0.0004
News	0.0190	0.0190	+0.10%	0.8244
Related Search	0.0211	0.0207	+1.93%	7e-26
Pagination	0.0226	0.0227	-0.44%	0.0114
Other	0.0518	0.0515	+0.58%	0.0048

Figure 17. Example Scorecard with metrics breakdown used in our case companies.

Scorecards with metric breakdowns enable experimenters to obtain an in-depth understanding of what actually happened in an OCE, and as a result make the feature teams more self-sufficient in evaluating experiment results. In addition, and by highlighting significant changes with colors, identifying a change and its direction is simplified.

Experimentation team organization. Experimentation team data scientists are embedded into individual feature teams to become domain experts and grow experimentation in the team. The benefit of this model is that the embedded experimenters can become experts in each business domain. The main disadvantage, however, is the lack of a clear career path for these professionals, who also may not receive peer feedback and mentoring that help them evolve.

Overall Evaluation Criteria (OEC). Evolved. OEC is validated over a large number of experiments and evolved with the findings from the learning experiments. For example, product teams should start running learning experiments where a small degradation in user experience is intentionally introduced for learning purposes (e.g. degradation of results, slowdown of a feature). With such learning experiments, teams will have a better understanding of the importance of certain features and the effect that changes have on the metrics. Knowingly hurting users slightly in the short-term (e.g., in a 2- week experiment) reveals fundamental issues and thereby improves the experience in the long-term. An OEC should become a single metric that consists as a weighted combination of other metrics (Data Quality, Success & Guardrails).

Experimentation Impact. Experimentation impacts infrastructure planning, product quality and product complexity, which are areas that typically span over multiple teams. For example, when a feature in a product is updated (e.g. an algorithm is optimized) companies risk introducing an effect on its underlying infrastructure. By conducting OCEs, practitioners can be more proactive and guarantee that (1) necessary capacity will be available for their changes, and (2) estimate the capacity cost/savings of the change before it is fully deployed. This can be achieved by measuring the

change in infrastructure load via an OCE on a small population, e.g. 10% vs 10%, and then project what will happen at 100% deployment. A gradual ramp-up of the experiment can be used to estimate non-linear effects. Also, and by experimenting with customers in a gradual manner, the impact of changes that introduce product issues is limited to only a defined number of customers. As a side benefit of this impact of OCE, the investments into virtual testing infrastructure mimicking production environment can be minimized as quality issues in OCEs will be constrained and detected very early.

8.5.2.4 Fly Stage

Technical focus of product development activities. In the previous phases, technical activities focused on implementing reliable instrumentation, creating comprehensive metrics and conducting learning experiments. This typically causes one-time metrics to be created, and many different types of scorecards to appear in various teams. In the ‘Fly’ stage, feature teams should be able to agree on a standard scorecard (for example, one scorecard that can be used by all of the feature teams). To manage this situation, and to preserve only the metrics and scorecards that are useful, those that are not active (for example, have not been viewed on a scorecard, are not used in experiment hypothesis) should become labelled for deletion. Similar consolidation should happen for data pipelines.

Experimentation platform capability. To conduct tens of thousands of OCEs yearly, experimentation platform should become Intelligent and support features such as *Interaction control and detection* (for example, detecting and correcting results when experiments interact with each other), and *near real-time detection and autonomous shutdown of harmful experiments*. In addition, and to enable experimenters to better understand the experiment outcomes, the experimentation platform should also support an accumulation of institutional memory and have drill-down capabilities to reveal reasons for metric movements. We present the features in greater detail next:

Interaction control and detection: A statistical interaction between two concurrently running treatments A and B exists if their combined effect is not the same as the sum of two individual treatment effects. This is a feature that prevents such experiments with conflicting outcomes to run on the same sets of users (e.g. one experiment is changing the background color to black, another the text to gray). Control for such interactions should be established and handled automatically. After detecting an interaction, the experimentation platform should send an alert to the experiment owners via email or tools like Slack to get their attention. Detailed guidance on how to implement this feature is available in [57].

Near real-time detection and autonomous shutdown. In this stage, and with many experiments testing many metrics, the naïve approach to alerting and reacting on every statistically significant negative metric change will lead to an unacceptable number of false alerts. The entire alerting system can become overloaded and hard to interpret for experimenters. To address this challenge, and before raising an alert and deciding on an action for it, the experimenters should take into consideration the following factors: (1) the type of the metric with the degradation, (2) the size of the change, and (3) the severity level.

- (1) Alerting should be configured on the success and data quality metrics. Alerting on various debugging metrics might be useful, however, at the same time take away the attention from the really important changes.
- (2) The difference between the treatment and the control should not only be statistically significant, but also large enough in absolute magnitude to have a meaningful impact. For example, an experimentation platform should not alert to a 1-millisecond degradation to Page-Load-Time metric, even if the confidence that the degradation is a real effect (e.g., the p-value is less than $1e-10$) is very high.
- (3) Different magnitudes of changes of experiment metrics are categorized in specific severity levels based on the number of customers impacted, and the amount of business degradation

(for example, revenue lost might have high severity than Page-Load-Time degradation).

Based on these three factors, the action that the experimenters take can be decided. For example, the most severe changes may result in a shutdown of an experiment. On the other hand, less severe changes could result in notifying the team over a communication tool (for example, Microsoft Teams or Slack) about its status.

The *accumulation of institutional memory*: The large number of experiments conducted in this stage by many individuals in several feature teams makes it increasingly more difficult to remember what has already been tested, based on what qualitative insight, and what the outcome of those OCEs was. To enable feature teams to benefit from the past experiment learnings, a mature experimentation platform should automatically capture *the qualitative insights* that led to the design of the experiment, *summary of the individual experiment*, the hypotheses that were evaluated, the *differences between the variants*, any *special instrumentation* or data that was captured specifically for the experiment and a definitive scorecard that was used to make the decision on. Most importantly, the decision *whether any of the variants were deployed* should be captured with qualitative insights that contributed to the decision. Capturing the learnings that we list above is critical for experimenters to get an overview of the OCEs that were conducted in individual areas (for example, relevance or performance), and to form better hypotheses for future experiments. In addition, information such as *links to any other scorecard* that was computed in an experiment and *a screenshot of the change* (in situations when the variations were visually different) may be helpful to keep.

Deep-Dive analysis: To make more informed decisions and discover what impacted a certain metric to change, an experimentation platform should enable top-down and bottom-up deep dive analysis of experiments. In a top-down scorecard analysis of an experiment, a scorecard that contains the aggregate data can be further broken down into *segment scorecards* (for example, one scorecard per browser type) and *time period scorecards* (for exam-

ple, a daily and a weekly scorecard). This type of breakdown enables experimenters to understand how homogeneous the treatment effect is across the different segments and time periods. Using this breakdown, experimenters might discover common heterogeneous effects in various segments (for example, a very old browser version might be causing a high number of errors when new technology is used to develop the site) and detect novelty and long-lasting effects (for example, the treatment group that received a new feature engages with it in the first day and stops using it the day after).

In addition, and to further narrow the scorecard to the most critical changes, an experimentation platform should enable hiding non-significant changes, or showing the significant changes under different thresholds (e.g. 0.05 or 0.001 significance thresholds). In a bottom-up scorecard analysis, experimenters are concerned in debugging how individual log signals affected the metric computation. For example, to reduce the noise in the log data when a change to a product is done on a checkout page, a deep-dive analysis might help in revealing the signal that triggers when users land on the page where the change was implemented. This signal can then be used (together with the counterfactual signal from the control variation) to provide experimenters with a triggered scorecard view where the test statistic is computed only on the affected (treatment) and ‘would be affected’ (control) users. This bottom-up analysis is also very useful in situations when debug metrics are not detailed enough to explain a change in a particular metric.

Experimentation pervasiveness. Contrary to previous stages where experimentation is seen as a tool to validate improvement hypotheses to software products, in this stage feature teams experiment with all changes, even bug fixes. In fact, many experiments in the ‘Fly’ stage could be no-harm experiments from the start. The view of experimentation is extended from value seeking activity to a safe-gate utility.

Feature team self-sufficiency. No Data Scientist involvement needed for most experiments. On top of the effortless experiment creation and management, experiment results should be intuitive

for feature teams to interpret and make decisions on. In experiments where additional help is needed from data scientists with experimental skills, feature teams contact a center of excellence (see below).

Experimentation team organization. Feature teams have their own experimentation experts that create a strong unity with business, and connect with a centralized experimentation team. On the other side, a centralized experimentation team serves as a center of excellence that partners with experimentation experts on feature teams in situations when help is needed. The primary focus of the center of excellence is, however, on improving the design, execution, and analysis of controlled experiments. A benefit of this organization is that by connecting the data scientists together, best practices can be spread throughout the organization and institutional memory shared in another way. This can be further accelerated by hosting classes for feature teams, labs, and creating internal conferences. The main disadvantage, however can be a lack of clarity about what the center of excellence owns and what the feature teams own, who should pay for hiring more data scientists when various feature teams grow the number of experiments, and who is responsible for investments in checks and alerts related to experimentation on the feature team's components.

Overall Evaluation Criteria (OEC). In this stage, the OEC is used to set the performance goals for teams within the organization, and monitor their progress in reaching them. In contrast to the previous phases where the OEC was evolving, changes to the overall evaluation criteria in the “Fly” phase should occur only periodically (e.g. once per year) and follow a standardized process. This gives independent teams across the product portfolio a chance to focus their work on understanding how the features they own affect the key metrics, prioritizing their work to improve the OEC.

Experimentation Impact. The impact of experimentation in this stage is widespread across the product. Teams working on the product are rewarded for experiment metric gains. At the same time, teams’ contributions are measured not only by the number of changes they deployed but also with the exact amount of metrics’ improvements. Impactful experiment results can be shared with top executives, board of directors, and externally. As illustrated in the empirical data section, experimentation and its impact may even be discussed on quarterly earnings calls. If experimentation growth starts with aligning the experimentation practices to the organization, in the ‘Fly’ stage the organization becomes the one that is aligned to it. Humble, however, very impactful.

8.5.3 Applying The Experimentation Growth Model

In the previous section, we described in greater detail our guidelines towards evolving experimentation practices. To benefit from our main contribution, the Experimentation Growth Model, software companies that are developing or growing their experimentation capabilities should first identify their current state of maturity for experimentation. By recognizing their current state, they can learn what to do in the short and long-term to grow their experimentation capabilities using the guidance presented in the Experimentation Growth Model and its evolutionary steps.

Step 1: Exp. Growth self-assessment: To enable companies an effortless and consistent self-assessment of their current state of experimentation capabilities based on our model, we developed an

online application available online at '<https://www.exp-growth.com>' [214]. The first two authors of this chapter designed the application in a form of a survey questionnaire. The survey was refined based on multiple rounds of feedback collection from test respondents (in total 54 people working at four of our case companies). Our questionnaire asks the responded 15 closed multiple-choice questions, and an open-ended question. Based on the responses, the application calculates an overall Experimentation Maturity score, and a score for every dimension of the Experimentation Growth model. At the same time, it visualizes the experimentation maturity assessment (see example on Figure 18 in yellow). In this way, software companies can, with very little effort, consistently evaluate the current state of their experimentation evolution.

Step 2: Exp. Growth evolving: By assessing their maturity for experimentation for each of the seven dimensions of the growth model, software companies can draw the roadmap for the necessary changes and investments that are needed to evolve and further develop their experimentation practices. For example, and as visible in Figure 6 from one of the anonymous software companies that assessed their current state of experimentation growth through our tool, this particular software company learned that they lack behind in 3 dimensions of experimentation growth. The company should 1) invest into standardizing their data collection practices and build informative metric groups, (2) form a centralized experimentation team, (3) and invest into designing and selecting metrics for their OEC.

Experimentation Growth Score

5.3

Your Experimentation Growth score is 5.3 out of 10.

You are well on the way to making full use of experimentation in your product! The table below shows the seven dimensions of experimentation growth and where you are along each dimension. Examine the table carefully and consider the aspects your product is missing. These are the things you need to focus on to advance further.

	Crawl	Walk	Run	Fly
Technical focus of product development activities	Data Collection Transitioning from the debugging logs to data collection practices that enable data-driven development, resolving data quality issues. Analysis of experiments is manual.	Standardization Standardizing data collection and processing. Creating a standard set of simple metrics: debug metrics (largest group), success metrics, guardrail metrics and data quality metrics.	Comprehensiveness Comprehensive data collection, reliable data pipeline, and an exhaustive set of metrics. Automated analysis of experiments.	Consolidation Unified data pipeline enables easy creation of cross-team metrics. Standardized process for metric creation, evaluation and management.
Experimentation platform capability	None An initial experiment can be coded manually (ad hoc). No experimentation platform needed.	Basic 3rd party platform can be used or internally developed. Basic features such as Trustworthy Randomization, Power Analysis, Pre-Experiment A/A testing are required.	Evolved The experimentation platform should be extended with new features supporting larger volume of experiments, such as Alerting, Control of carry-over effects, Experiment iteration support.	Advanced Intelligent platform supporting features such as Interaction control and detection, Near real-time detection and automatic shutdown of harmful experiments, Drill-down capabilities to understand metric movement.
Experimentation pervasiveness	Generating management support A handful of experiments to showcase value of experimentation and generate management support (e.g. design options).	Experimenting with a few features Broadening the types of experiments on a selected set of features (e.g. from design to performance, from performance to infrastructure).	Expanding to more features Experiment on most new features.	Experimenting with every change Experiment with all changes, even bug fixes.
Engineering team self-sufficiency	Limited Feature team relies on a Data Scientist to setup, manage and analyze an experiment.	Experiment Setup Creating the experiment (instrumentation, A/A testing, assigning traffic) is managed by the feature team.	Experiment Management Feature team is responsible for monitoring experiments, making ramp-up and shut-down decisions, designing and deploying experiment-specific metrics.	Experiment Analysis No Data Scientist involvement needed for most experiments. Experiment results are intuitive for feature team to interpret and make decisions.
Experimentation team organization	None No experimentation team.	Standalone Centralized experimentation team is created. No or very little experimentation skills in feature teams.	Embedded Experimentation team data scientists are embedded into individual feature teams to become domain experts and grow experimentation in the team.	Partnership Feature teams have their own experimentation experts that create a strong unity with business. Experimentation team serves as a center of excellence partnering with experimentation experts on feature teams.
Overall Evaluation Criteria (OEC)	Experiment-specific OEC is defined in a custom way for every experiment.	Defined OEC is defined as a structured set of metrics consisting of Success, Guardrail and Data Quality metrics.	Evolved OEC is validated over a large number of experiments and evolved with the findings from the learning experiments. Single metric as a weighted combination of others is desired.	Stable OEC is stable, only periodic changes are allowed (e.g. 1 per year). It is used to set the performance goals for teams within the organization.
Experimentation impact	Feature Constrained to the incremental feature update within the team that is starting to experiment.	Team Starting to impact the team's activity planning (e.g. release cycle). The impact is shared within the larger team.	Multiple teams Experimentation impacts infrastructure planning, product quality and product complexity (e.g. controlled feature removal).	Executives and Externally The impact of experimentation is widespread across the product. Product teams are rewarded for experiment metric gains. Experiment results are shared with top executives, Board of Directors, and externally.

Figure 18. Experimentation Growth Maturity example.

<http://www.exp-growth.com/>

8.5.4 The Challenging Side of Experimentation

Experimentation in software product development is frequently presented as a magic pill that cures many diseases ingrained in a software product. Our prior work is no exception to this [9], [25]. And although we did not conduct a thorough analysis of our empirical data for the purpose to identify all of the current challenges in experimentation, we wish to contrast our overall positive view by presenting two concerns of experimentation that appeared in our discussions most frequently. The limitation with the low number of users and a concern with the ethics of experimentation.

Randomization unit count: Not specific to experimentation, however in the essence of it, is the size of the population that companies can sample from and randomize on (their customer base). To measure very sensitive changes to metrics, many randomization units are needed in the experiment variations. For example, to measure a subtle change to a key OEC metric, several millions of users might be needed over a period of a few weeks to get the statistical power that is required to detect the desired effect. Although there are a number of different ways to tackle this problem (for example, to run experiments longer, to reduce noise in the data by analyzing only the units that were impacted by the change - known as triggered analysis, to make larger changes, etc.) many of them suffer from pitfalls that need to be carefully controlled for [15], [16], [187]. However, one way to really improve the accuracy of the analysis is to increase the number of randomization units assigned to an experiment. Assuming that companies have enough units (for example users), assigning more units to one experiment may block other experiments from starting, creating queues of experiment variations that are waiting to be tested. In the scenario when this assumption is not true, a controlled experiment may not be helpful. In fact, for one of the case companies that we work with, experimentation is an activity that can be used effectively only during the periods when many customers use their solution. Such limitations give the accumulation of OCEs learnings an ever-

higher priority, and prioritization of experiment execution a new meaning.

Ethics of experimentation: In some scenarios, OCEs can have a profound impact on the users that are exposed to different experiences. For example, a software company might have an interest in tricking users into converting to something that they might not want to, removing features from some of their product users, or even changing how their customer base behaves. To provide a concrete example, we reference an experiment from Facebook [144] where positive news were suppressed from the treatment group, causing the users of Facebook in this group to express aggressive negative opinions on the social network site. Experiments that change user behavior to similar extent can be considered unethical, and software companies should take this into consideration when (1) designing experiments and (2) deciding in what way to use their results. We support this with an illustrative quote from one of our case companies:

“Some experiments don’t ship because they are not good for users and impacting trust of users would make the cost of learning too high. Trust is the most important thing for us, that’s why we often make decisions against metrics and rely on our community to naturally police things”

--Case Company J

Without doubt, more discussion and research is needed to address the two considerations above, and to enable even more transparent and efficient experimentation for software companies with customers using their products around the globe.

8.6 Conclusion

Learning from data has always been critical for software companies [6], [28]–[31]. More than ever, companies today realize the importance of using the data to reveal how their customers actually use their products [23], [133], [175], [198], in addition to data that reveals what customers say they do with these products [59], [64], [215], [216]

At a time when online services are becoming vital to almost all businesses, rigorous online controlled experiments should be standard operating procedure in companies with many customers [14], [58], [189]. If a company develops the software infrastructure and organizational skills to conduct them, it will be able to assess not only product ideas, but also potential business models, strategies, products, services, and marketing campaigns—all relatively inexpensively [11]. To achieve this level of maturity for experimentation, however, companies need to grow and evolve their experimentation capabilities.

In this chapter, and based on case study research conducted with four large-scale software companies that are in the forefront of experimentation capability, we present the seven most critical aspects of experimentation, and explain in greater detail how to address them in each of the four stages of maturity for experimentation. Our main result is the Experimentation Growth Model, which extends and redefines our prior work. Specifically, we redefined previously recognized dimensions, extended our work with a new dimension of experimentation evolution (the impact of experimentation), we showcased learnings through examples of online controlled experiments from four large scale case companies, and described individual stages of experimentation in much greater detail. With our work, we aim to help other software companies in evolving their experimentation practices.

9 EXPERIMENTATION PLATFORM

Abstract

Online controlled experiments (e.g., A/B tests) are an integral part of successful data-driven companies. At Microsoft, supporting experimentation poses a unique challenge due to the wide variety of products being developed, along with the fact that experimentation capabilities had to be added to existing, mature products with codebases that go back decades. This chapter describes the Microsoft ExP Platform (ExP for short) which enables trustworthy A/B experimentation at products from web properties (such as bing.com) to mobile apps to device drivers within the Windows operating system. The two core tenets of the platform are *trustworthiness* (an experiment is meaningful only if its results can be trusted) and *scalability* (we aspire to expose every single change in any product through an A/B experiment). Currently, over ten thousand experiments are run annually. In this chapter, we describe the four core components of an A/B experimentation system: experimentation portal, experiment execution service, log processing service and analysis service, and explain the reasoning behind the design choices. These four components provide a system where ideas can turn into experiments within minutes and experiments can provide trustworthy results within hours.

This Chapter is based on

*S. Gupta, S. Bhardwaj, P. Dmitriev, U. Lucy, A. Fabijan, and P. Raff, “The Anatomy of a Large-Scale Online Experimentation Platform,” to appear in **Proceedings**¹ of IEEE International Conf. on Software Arch., ICSA'18, (Seattle, USA, May 2018).*

¹ © 2018 IEEE. With permission from the authors.

9.1 INTRODUCTION

Online controlled experiments (e.g., A/B tests) are becoming the gold standard for evaluating improvements in software systems [29]. From front-end user-interface changes to backend algorithms, from search engines (e.g., Google, Bing, Yahoo!) to retailers (e.g., Amazon, eBay, Etsy) to social networking services (e.g., Facebook, LinkedIn, Twitter) to travel services (e.g., Expedia, Airbnb, Booking.com) to many startups, online controlled experiments are now utilized to make data-driven decisions at a wide range of companies [11]. While the theory of a controlled experiment is simple, and dates to Sir Ronald A. Fisher' [129], the deployment of online controlled experiments at scale (for example, 100's of concurrently running experiments) across variety of web sites, mobile apps, and desktop applications is a formidable challenge.

Consider the following experiment that ran in Skype mobile and desktop apps. When the user attempts a call but the caller does not answer, a dialog is shown prompting the user to leave a video message.

Many steps need to happen to execute this experiment. First, the code that shows the message needs to be deployed to the clients in such a way that it can be turned on and off via the experimentation system. Then, the audience, the experiment design, the experiment steps, and the size and duration for each step need to be determined. During the experiment execution, correct configurations need to be delivered to the users' mobile and desktop apps, ideally in a gradual manner, while verifying that the experiment is not causing unintended harm (e.g. app crashes). Such monitoring should continue throughout the experiment, checking for variety of issues, including interactions with other concurrently running experiments. During the experiment, various actions could be suggested to the experiment owner (the person who executes the experiment), such as stopping the experiment if harm is detected, looking at surprising metric movements, or examining a specific user segment that behaves differently from others. After the exper-

iment, results need to be analyzed to make a ship/no-ship recommendation and learn about the impact of the feature on users and business, to inform future experiments.

The role of the experimentation platform in the process described above is critical. The platform needs to support the steps above in a way that allows not only a data scientist, but any engineer or product manager to successfully experiment. The stakes are high. Recommending an incorrect experiment size may result in not being able to achieve statistical significance, therefore wasting release cycle time and resources. Inability to detect harm and alert the experiment owner may result in bad user experience, leading to lost revenue and user abandonment. Inability to detect interactions with other experiments may lead to wrong conclusions.

At Microsoft, the experimentation system (ExP) [217] supports trustworthy experimentation in various products, including Bing, MSN, Cortana, Skype, Office, xBox, Edge, Visual Studio, Windows OS. Thousands use ExP every month to execute and analyze experiments, totaling over ten thousand experiments a year.

In this chapter, we describe the architecture of ExP that enables such a large volume of diverse experiments, explain the reasoning behind the design choices made, and discuss the alternatives where possible. While the statistical foundations of online experimentation and their advancement is an active research area [26], [218], and some specific aspects of experiment execution such as enabling overlapping experiments [58] were discussed elsewhere, to our knowledge this is the first work focusing on the end-to-end architecture of a software system supporting diverse experimentation scenarios at scale.

The two core tenets of ExP are *trustworthiness* and *scalability*. “Getting numbers is easy; getting numbers you can trust is hard” [219]. Producing trustworthy results requires solid statistical foundation as well as seamless integration, monitoring, and variety of quality checks of different components of the system. Scalability refers to the ability of a new product or team to easily onboard and start running trustworthy experiments on ExP at low cost, and then enable more and more ExP features as the experimentation

volume grows, until every single product feature or bug fix, as long as it is ethical and technically feasible, is evaluated via an experiment [9].

The four core components of ExP are *experimentation portal*, *experiment execution service*, *log processing service*, and *analysis service*. Experimentation portal is an interface between the experiment owner and the experimentation system, enabling the owner to configure, start, monitor and control the experiment throughout its lifecycle (see Figure 1). Experiment execution service is concerned with how different experiment types are executed. Log processing service is responsible for collecting and cooking the log data and executing various analysis jobs. Analysis service is used throughout the whole experiment lifecycle, informing experiment design, determining parameters of experiment execution, and helping experiment owners interpret the results.

The contribution of our chapter is threefold: We (1) describe the architecture of ExP, Microsoft's large scale online experimentation platform, supporting experiments across web sites, mobile and desktop apps, gaming consoles, and operating systems. We (2) discuss choices and alternatives, which, in other scenarios and for other companies, may potentially be better options when building their own experimentation platform. We (3) illustrate the discussion with examples of real experiments that were ran at Microsoft.

With increasing use of experimentation in software development, more and more companies are striving to expand their experimentation capabilities. We hope that this chapter can serve as a guide, helping them build trustworthy and scalable experimentation solutions, ultimately leading to more data driven decisions benefitting the products and their users.

9.2 BACKGROUND & RESEARCH METHOD

9.2.1 Online Controlled Experiments

In the simplest controlled experiment or A/B test users are randomly assigned to one of the two variants: control (typically denoted as A) or treatment (typically denoted as B). Elsewhere in this chapter, we will use V1, V2, V3... to represent variants, and an experiment is defined as a set of variants, e.g. $E = \{V1, V2\}$. Usually control is the existing system, and treatment is the existing system with a change X (for example, a new feature). While the experiment is running, user interactions with the system are recorded, and metrics are computed. If the experiment were designed and executed correctly, the only thing consistently different between the two variants is the change X. External factors are distributed evenly between control and treatment and therefore do not impact the results of the experiment. Hence any difference in metrics between the two groups must be due to the change X (or a random chance, that we rule out using statistical testing). This establishes a causal relationship between the change made to the product and changes in user behavior, which is the key reason for widespread use of controlled experiments for evaluating new features in software. For the discussion that follows, an important aspect is where an experiment is executed. If the experiment is executed by the user's mobile/desktop app, gaming console, OS, etc. we label it a *client-side experiment*. If the experiment is executed by the web site, call routing infrastructure, or other server component, we label it a *server-side experiment*.

9.2.2 Related Work

Controlled experiments are an active research area, fueled by the growing importance of online experimentation in the software industry. Research has been focused on topics such as new statistical methods to improve metric sensitivity [188], metric design and interpretation [44], [122], projections of results from a short-term experiment to the long term [15], [184], the benefits of experimentation at scale [11], [25], experiments in social networks [37], as well as examples, pitfalls, rules of thumb and lessons learned from running controlled experiments in practical settings [154], [192], [16]. These works provide good context for our chapter and highlight the importance of having a solid engineering platform on top of which the new research ideas, such as the ones referenced, can be implemented and evaluated. Our work describes how to build such a platform.

High-level architecture of an experimentation system was discussed in section 4.1 of [57] and partially in [9], [135]. The discussion, however, is very brief, and, most importantly, only concerns experimentation on web properties. Similarly this is the case in [58]. Supporting client experimentation, such as desktop or mobile apps, requires a different architecture. Moreover, since [57] and [58] were published, new experiment design and execution approaches were developed, e.g. [188], which also require a different architecture to support them. Therefore, to introduce a detailed architectural design of an experimentation platform that supports the aforementioned experimentation scenarios, we conducted an in-depth case study at Microsoft, which we briefly describe next.

9.2.3 Research Method

The result of this chapter builds on case study research [137] conducted at Microsoft Corporation – Case Company H.

Case Company. Microsoft is a multinational organization and employs over 120 000 people. Most of the authors of this chapter have been working at the case company for several years with many product teams (e.g. Bing, Skype, Office, XBOX etc.) specifi-

cally focusing on enabling and advancing their experimentation capabilities.

Data collection. The authors of this chapter have good access to the meeting notes, design documents, notebooks and the experimentation platform itself. We collected and placed all existing documents that describe the ExP in a shared folder, and complemented them with a joint notebook with personal notes and insights about ExP that were previously not documented.

Data Analysis. We aggregated the collected data describing the ExP platform, and jointly triangulated the historical data describing the reasoning behind the various architectural decisions of the ExP platform with our recent and extensive experience. In principal, we applied the thematic coding approach where we grouped similar concepts under categories, and aggregated these further until we agreed on a level of abstraction. In this way, for example, we emerged with “Experiment Execution Service” label and other components that describe our platform. During the analysis process, several people working at the case company reviewed our work and provided feedback on it.

Threats to Validity. *Construct validity:* Our case company has a long tradition of experimentation research and applying it in practice. The authors of this chapter and others that contributed to it are very familiar with the architecture of the ExP platform, and the needs and challenges that it has to address. *External Validity:* The discussion in this chapter and its main contribution is based on a system implemented at a single company. This is a limitation that we acknowledge. Despite this concern, however, we believe that the diverse range of products that our platform supports make our contributions generalizable to many other scenarios across the software industry, including to companies developing web sites, mobile and desktop apps, gaming consoles, services, and operating systems.

9.3 ExP PLATFORM ARCHITECTURE

The ExP platform aims to facilitate the *experimentation lifecycle* (see Figure 1), which is the basis of data-driven product development at Microsoft.

9.3.1 Experimentation Lifecycle

All product changes (e.g. new features) should start with a *hypothesis*, which helps explain the reasoning for the change. Good hypotheses originate from data analysis and should be quantifiable. In the Skype experiment example introduced in Section 1, the hypothesis could be that *at least 1 out of 10 of users who had an unanswered call would leave a video message*, leading to an increase in the number of calls (due to more returned calls) and an increase in the overall use of Skype.

Figure 19. The experimentation lifecycle.

The hypothesis then goes through the steps of implementation and evaluation via an experiment. At the end of the experiment, the decision is made to ship the change or abort. Regardless of the outcome, the learnings from the experiment become the basis of a new hypothesis, either an incremental iteration on the current feature, or something radically different.

In the remainder of this section, we discuss the end-to-end architecture of the experimentation system, which we also depict on Figure 3 on the next page.

9.3.2 Experimentation Portal

Experimentation portal serves as an interface between the experiment owner and the experimentation system. The goal of the experimentation portal is to make it easy for the experiment owner to create, configure, run, and analyze experiments. For experimentation to scale, we need to keep making these tasks easier.

The three components of the experimentation portal are *experiment management*, *metric definition and validation*, and *result visualization and exploration*.

Figure 20. An iteration of the Bing logo experiment.

a) *EXPERIMENT MANAGEMENT*: Consider the following example of an experiment ran in Bing, aimed to understand the impact of changing the Bing’s logo, as shown in Figure 2. We will use this example to illustrate the key choices the experiment owner has to make when creating and managing an experiment.

Audience often depends on the design of the feature. For the example in Figure 2, since this is an English language logo, only users in the United States-English market were exposed to the experiment. It is a common practice to hone your feature and learnings in

one audience, before proceeding to expand to other audiences. The experimentation portal needs to provide flexible audience selection mechanisms, also known as *targeting* or *traffic filters*. Examples of simple traffic filters are market, browser, operating system, version of the mobile/desktop app. More complex targeting criteria may involve prior usage information, e.g. “new users” who started using the app within the last month.

Figure 21. Architecture of the experimentation platform.

b) **Overall Evaluation Criteria (OEC)** is, essentially, the formalization of the success of experiment hypothesis in terms of the expected impact on key product metrics. It is important that the OEC is determined before the experiment starts, to avoid the temptation of tweaking the ship criteria to match the results, and because it serves as input for determining the experiment size and duration. Ideally, most experiments should optimize for a single product-wide OEC rather than individual experiment-specific OECs [9]. For the example in Figure 2, the OEC was the standard Bing OEC discussed in [220]. The experimentation portal should require the experiment owner to formally document the OEC when creating an experiment.

c) **Size and duration.** The experiment owner needs to specify what fraction of the targeted population will receive the treatment and the control, and **how long** the experiment should run. The key tradeoff here is that the experiment owner would be risk averse and would want to exposure as few users as possible to the new feature but still be able to detect sizable changes in key metrics. Power analysis [205] helps in making this decision. Do note that running an experiment longer is not always better in terms of detecting small changes in metrics [220]. One key input in power analysis is the *coverage* of the new feature: the proportion of users in treatment who will experience the new feature. In cases of small coverage triggered analysis can help improve the power of metrics [18]. Other factors like apriori expectation of novelty or primacy effects in features, or warm up period needed for machine learning models may also be a factor for determining the duration of an experiment. It is important to stick to the duration of the experiment. Stopping the experiment early or increasing the duration after looking at the results can lead to wrong conclusions [16].

d) **Experiment template.** While the basic experiment setup described in Section II.A can be used, it may suffer from *random imbalance*. When the users are randomized into variants, just by chance there may be an imbalance resulting in a statistically significant difference in a metric of interest between the two populations. Disentangling the impact of such random imbalance from the impact of the treatment may be hard. Following the idea of [221], ExP supports several types of experiments that use *re-randomization* to alleviate this problem. The naïve way to detect random imbalance is to start an experiment as an A/A (treatment and control are the same in this stage), run it for several days, and if no imbalance is detected, apply the treatment effect to one of the A's. If random imbalance is present, however, the experiment needs to be restarted with a re-randomization, resulting in a delay. When this experiment type was used in Bing about 1 out of 4 experiments had to be restarted. Another option is to perform a *retrospective A/A analysis* on historical data, the *Pre-Experiment Analysis* step in Figure 3. For example, we can take the historical

data for the last one week and simulate the randomization on that data using the hash seed selected for the A/B experiment about to start, as if a real A/A experiment ran during that time, and check for imbalance. Expanding on the above idea, we can perform many (e.g. 1000) retrospective A/A steps simultaneously and then select the most balanced randomization hash seed to use for the A/B step. We call this experiment type *SeedFinder*, referring to its ability to find a good hash seed for randomization. To implement SeedFinder, the experimentation portal needs to interact with the analysis service even before the experiment has started, and the analysis service needs to be able to precisely simulate offline the targeting and randomization as if it happened in the online system. Another aspect of experiment configuration is *ramp-up*: the sequence of steps the experiment will take to reach its final size. Ramp-up is particularly important for risky experiments, where the experiment owner may want to start them at a very small size and gradually move to the final desired size. ExP platform encapsulates different options for experiment types and ramp-up scheduled into experiment templates. Each product has its own set of templates that may include, in addition to the aspects described above, choices for what randomization unit to use in the experiment and alternative experiment designs such as interleaving [222]. Each product has a default recommended template.

e) **Experiment Interactions.** When many experiments configured as discussed earlier in this section are running concurrently, every user has a chance to be in any subset of these experiments at the same time. While often it does not create any issues, it is problematic in cases where interactions occur between variants of different experiments. Consider a hypothetical example where one experiment changes the font color on a web page to blue, while another changes the background color to blue. A mechanism to prevent the user from being in both of these experiments at the same time is needed. To address the need to execute multiple potentially interacting experiments concurrently, we use the notion of *isolation group*. Each variant V is tagged with one or more isolation groups,

denoted by $IG(V)$. Isolation groups must satisfy the *Isolation Group Principle*: if $V1$ and $V2$ are two variants and $IG(V1) \cap IG(V2) \neq \emptyset$, then a user can never be concurrently assigned to both $V1$ and $V2$. To avoid assignment mismatches between the variants, we require all variants in the experiment to share the same set of isolation groups. The problem of taking a global set of variants – each with its own set of isolation groups – and providing a set of assignments for each user will be discussed in Section III.C.

f) **Variant behavior.** Experiment owner needs to configure the behavior for each variant. For server-side experiments, where the product code can be updated at any time, configuration management system such as the one described in [213] can be used to manage large amount of configurations. For client-side experiments, where the software is only updated periodically, an online configuration system needs to be used to turn the features of the product on and off. Discussion of the design of configuration systems is beyond the scope of this chapter.

2) METRIC DEFINITION AND VALIDATION:

To obtain accurate and comprehensive understanding of experiment results, a rich set of metrics is needed [16]. Therefore, to scale experimentation it is important to provide an easy way to define and validate new metrics, that anyone in the company is able to use. The process of adding new metrics needs to be lightweight, so that it does not introduce delays into the experiment life cycle. It is also important to ensure that metric definitions are consistent across different experiments and business reports.

To satisfy the above requirements, we developed a Metric Definition Language (MDL) that provides a formal programming language-independent way to define all common types of metrics. The user-provided metric definition gets translated into an internal representation, which can then be compiled into a number of target languages, such as SQL. Having a language-independent representation is particularly useful, because different Microsoft products use different backend data processing mechanisms, and sometimes the same product may use several mechanisms. MDL ensures that

the definition of a metric is consistent across all such pipelines. See Figure 4 for an example of MDL.

Experimentation portal provides a user interface enabling everyone to easily create and edit MDL metric definitions, test them on real data, and deploy to the Metric Repository. This repository is then used by all experiment analysis and business reporting jobs. On average, over 100 metrics are created or edited every day using the metric definition service.

Figure 22. MDL representation of the AvgClicksPerUser metric: definition (top), internal representation (middle) and auto-generated SQL (bottom).

4) RESULT VISUALIZATION/EXPLORATION:

While the experiment is running, and after its completion, various analysis results are generated: quick near-real time checks are performed, large experiment scorecards with thousands of metrics are computed, alerts are fired, deep-dive analyses are automatically kicked off, and manually configured custom analyses are created. The goal of the Results Visualization and Exploration component is to serve as a single place where experiment analysis information

is collected, providing visualizations and tools to help the experiment owner explore the results learning as much as possible, and make the correct ship/no-ship decision. The portal also allows configuring and submitting additional analysis requests.

The simplest form of experiment results is an experiment scorecard – a table consisting of a set of metrics and their movements in an experiment. In a typical scorecard comparing two variants, control and treatment, the following minimal information should be displayed for every metric:

- The observed metric values for control and treatment.
- The delta between these observed values, both in absolute and relative terms.
- The results of a statistical test to convey significance of the delta, typically via a p -value from a t-test.

At Microsoft, it is common for experiment scorecards to compute thousands of metrics over dozens of population segments, making manual scorecard analysis tedious and error prone. Important insights can be missed, especially if they are not expected by the experiment owner or heterogeneity is present in the treatment effect [223], and a large number of metrics gives rise to the multiple comparisons problem [224]. ExP provides a number of visualizations and analysis tools to automatically identify and highlight interesting insights for the experiment owner, and help avoid pitfalls such as the ones mentioned above. Some of these solutions are discussed in [16].

In addition to helping configure, run, and analyze experiments, Experimentation Portal also acts as a central repository of all previously run experiments, collecting metadata about each experiment and its results, and providing anyone an opportunity to learn about the past successes and failures, helping to disseminate learnings from experiments across the company.

9.3.3 Experiment Execution Service

The goal of the experiment execution service is generating variant assignments and delivering them to the product.

1) GENERATING VARIANT ASSIGNMENTS

At any given point in time, the system has its global experimentation state, which consists of the set of active variants, along with the following information about each variant:

- The set of isolation groups.
- The traffic filters.
- The percentage of total traffic configured.

0	1	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

Figure 23. Diagram of a numberline for a single isolation group.
A user who hashes into bucket 7 will always hash into bucket 7.

0	1	2	3	4	5	6	7	8	9
Treatment				Control					

Figure 24. Diagram of a numberline where buckets 0-3 have been allocated to treatment and 4-7 to control.

An assignment request comes with a certain randomization unit (e.g. user id) and as well as other properties that allow resolving the traffic filters. Given this information, the goal of the experiment assignment service is to compute the set of variants for this request.

Let us first consider the scenario of one isolation group. The isolation group has an associated numberline, which can be represented as a sequence of numbers (called buckets), with each bucket representing an equal portion of the population, and the whole representing 100% of the population. For any numberline, each user is consistently hashed into one specific bucket based on their user id, using a hash function such as [225]. For example, Figure 5 shows a numberline with ten buckets, where a user has been assigned to bucket 7.

An experiment that utilizes this isolation group will be allotted a set of buckets, as determined by availability and granularity of the numberline. If an experiment were to utilize this isolation group and consisted of a 40% treatment and a 40% control, then buckets

0-3 could be allotted to treatment and 4-7 to control. In this case, the user who hashes into bucket 7 would be assigned to control for this experiment. This is demonstrated in Figure 6.

Now that assignment for an individual isolation group has been established, we can provide assignment globally by instituting a priority order on isolation groups. Each experiment will utilize the numberline associated with the highest-priority isolation group involved as long as all other isolation groups for that experiment are still available. Then we can iterate through the isolation groups in their priority order, assign on each isolation group, and remove from consideration lower-priority isolation groups that were part of the assigned variant.

2) DELIVERING VARIANT ASSIGNMENT

There are three ways in which a client can obtain a treatment assignment, depicted and compared in Figure 7.

a) Via a service call. In this simple architecture, the client/server is responsible for explicitly making a service call to the experiment assignment service to obtain treatment assignments. The advantage of this architecture is that it applies equally to server-side and client-side experiments. The disadvantage is that the service call introduces a delay. Because of this care should be taken to select the appropriate time to make the service call. The typical pattern is to make the call as soon as the application starts (in order to allow experimenting on first run experience), and then periodically, avoiding doing it during important user interactions (e.g. in the middle of a Skype call). Often the application may choose to delay applying the assignment until it restarts, to avoid changing the user experience in the middle of the session.

b) Via a request annotation. In this architecture, the service is situated behind an Edge network (the network of servers responsible primarily for load balancing) and treatment assignment is obtained as an enrichment when the user's request passes through the Edge network. The service then obtains the treatment assignment from that enrichment, typically via added HTTP headers. This is a

common architecture for web sites because it does not require making an explicit service call. This architecture, however, is not applicable to client experimentation scenarios, because user interaction with clients are often local not requiring remote requests (e.g. editing a document on a PC.)

c) Via a local library. In this architecture, the client utilizes solely local resources for treatment assignment. The local library relies on configuration that is periodically refreshed by making a call to the central variant assignment service. This architecture is the best for client experimentation scenarios, and applicable to server experimentation scenarios as well. An application does not need to deal with making service calls explicitly, and also does not need to pass the treatment assignments around the stack as every component in the stack can call the library as often as needed without incurring the delay penalty.

Delivery Type	Architecture	Pros	Cons
Via a service call		Simple and applicable to both server and client experimentation scenarios.	Service call incurs a delay
Via a request annotation		No need to make a service request and incur delay penalty.	User request needs to pass through the Edge network, limiting applicability to mostly web sites.
Via a local library		No need to make a service request. Applicable to both client and server scenarios.	More complex implementation compared to the other approaches.

Figure 25. Comparison of different variant assignment delivery mechanisms.

3) CONFIGURATION

In addition to obtaining its variant assignment, the client also needs to know how to change its behavior based on these assignments. For server-side experiments, where the code can be updated any time, these variant configurations can be shipped as part of the code. For client-side experiments, updating the code may be a lengthy process, and a typical approach is to ship features dark (turned-off), and then utilize a configuration system to turn them on and off in response to different variant assignments. While variant assignment and configuration services have different goals, it is convenient to collocate them so that configurations are delivered to clients along with variant assignments.

9.3.4 Log Processing Service

The job of the log processing service is to collect the different types of log data (e.g. unique identifiers, timestamps, performance signals, context flags, etc.) and then merge, enrich, and join it to make it easier to consume by analysis jobs, the processed we call cooking. The data then becomes available for consumption by different data processing systems. Near real time processing system typically operates on small amounts of recent raw or lightly cooked data and is used to perform time-sensitive computations such as real-time alerts. Batch processing system operates on large volume of data and is used to perform expensive operations, such as com-

puting a scorecard with thousands of metrics over the whole experiment period.

1) DATA COLLECTION

From experiment analysis point of view, the goal of data collection is to be able to accurately reconstruct user behavior. During experiment analysis, we will both measure different user behavior patterns, and interpret them as successes and failures, to determine whether the change improves or degrades user experience. In data driven products that make extensive use of experimentation, logging the right data is an integral part of development process, and the code that does it adheres to the same standards (design, unit testing, etc.) as the code that implements core product functionality.

An important type of telemetry data is counterfactual logs. Having such signal allows for triggered analysis, greatly improving the sensitivity of analysis [18].

Telemetry data is usually collected from both the client and the server. Whenever there is a choice of collecting the same type of data from the client or from the server, it is preferable to collect server-side data. Server-side telemetry is easier to update, and the data is usually more complete and has less delay.

2) DATA COOKING

The goal of data cooking is to merge and transform the raw logs from all the different sources into the form that is easy and efficient to consume by analysis jobs. For example, when a web page is shown to the user there's a record about it in the server logs. When the user clicks on a link on the page, there's a record for it in the client logs. During cooking these two records will be merged making it much easier to compute metrics such as click-through rate.

Since telemetry keeps evolving over time, the cooking pipeline needs to be designed in a way that easily accommodates new telemetry events and changes to the existing events. Ideally, for most

common telemetry changes, no manual changes to the pipeline should be needed.

An important property of the cooking pipeline is the “No data left behind” principle. It means that any event or event property that existed in the raw logs should, in one form or another, be present in the cooked log. This property is essential both for ensuring the accuracy of analysis results, and for deep dive investigations of the reasons for metric movements.

As discussed in section 4.3.1, data delay and loss are common in client experimentation, and it is the job of the cooking pipeline to handle correctly delayed and incomplete data, update the cooked data as delayed events come in, and enrich the cooked data with extra signals for measuring the rate of data loss.

3) DATA QUALITY MONITORING

A critical piece of log processing service is data quality monitoring. Our experience is that intentional and unintentional changes to telemetry and cooking process happen all the time, impacting the correctness of the experimentation metrics. If such changes go unnoticed by experiment owners, metrics may be computed incorrectly, potentially resulting in wrong decisions on experiments.

Typical properties of the data to monitor are volume, fractions of outlier, empty, and null values, and various invariants – the statements about the data that we expect to always hold. An example of an invariant is “every Skype call with the status `Connected=true` should have a non-zero duration”, and conversely “every Skype call with the status `Connected=false` should have zero duration”. Once the cooked data stream is produced, a separate job needs to run to compute various properties of the stream, monitoring conditions should be evaluated, and alerts triggered if any issues are detected.

9.3.5 Experiment Analysis

1) ANALYSIS LIFECYCLE

Experiment analysis is conducted during the entire lifecycle of an experiment – hypothesis generation, experiment design, experiment execution, and post experiment during the decision-making process.

a) Hypothesis Generation. Results of historical experiments inform new hypotheses, help estimate how likely the new hypothesis is to impact the OEC, and help prioritize existing ideas. During this stage, the experiment owner examines other historical experiments similar to the one he/she is planning to run, as well as historical experiments that improved the metrics he/she is hoping to improve.

b) Experiment Design. During experiment design, analysis is performed to answer the following key questions:

- What kind of randomization scheme to use?
- How long should the experiment run?
- What percentage of traffic should be allotted?
- What randomization seed to use to minimize the imbalance?

c) Experiment Execution. While the experiment is running, the analysis must answer two key questions:

- Is the experiment causing an unacceptable harm to users?
- Are there any data quality issues yielding untrustworthy experiment results?

d) Post-Experiment. At the end of the experiment, to decide on the next steps, analysis should answer the following questions:

- Is the data from the experiment trustworthy?
- Did the treatment perform better than control?
- Did the treatment cause unacceptable harm to other key product metrics?
- Why did the treatment do better/worse than control?
- Are there any heterogeneous effects, with treatment impacting different populations differently?

To answer these and other analysis questions analysis service runs a number of automated and manual analysis jobs, generates notifications and alerts for the experiment owners, and provides tools for deep dive analysis and debugging. We discuss several key analysis topics in more detail next.

2) METRIC COMPUTATION

Care needs to be taken to compute the variance correctly during the metric computation. While the variance of the count and sum metrics computed at the randomization unit level is straightforward, percentile metrics and metrics computed at other levels where units are not independent (e.g. average Page Load Time computed as sum of page load times over the number of pages loaded across all users) require the use of delta method and bootstrap [26]. Wrong variance calculation may lead to incorrect p-values and potentially incorrect conclusions about the experiment outcome. In [188] a technique was introduced that uses pre-experiment data to reduce the variance in experimentation metrics. We found that this technique is very useful in practice and is well worth the extra computation time it requires, because in large products very small changes in the metric can have millions of dollars impact on annual revenue.

Another challenge with metric computation is computing large number of metrics efficiently at scale. With thousands of scorecards computed every day, each often requiring to process terabytes of data and perform sophisticated computations such as the one mentioned above, this is a non-trivial challenge. We have found that caching computations that are common across multiple experiments is helpful for reducing data size and performance.

3) SCORECARD ORGANISATION

While the naïve view of the scorecard as an arbitrary organized set of metrics may work for products that are just starting to run experiments, the number of metrics grows quickly, with most products using hundreds to thousands of metrics. This makes examining the metrics table manually very time consuming. Several scorecard organization principles and analysis components can help make it easier for users to understand the results and highlight important information they should not miss.

Metric	T	C	Delta (%)	p-value
Overall Click Rate	0.9206	0.9219	-0.14%	8e-11
Web Results	0.5743	0.5800	-0.98%	~0
Answers	0.1913	0.1901	+0.63%	5e-24
Image	0.0262	0.0261	+0.38%	0.1112
Video	0.0280	0.0278	+0.72%	0.0004
News	0.0190	0.0190	+0.10%	0.8244
Related Search	0.0211	0.0207	+1.93%	7e-26
Pagination	0.0226	0.0227	-0.44%	0.0114
Other	0.0518	0.0515	+0.58%	0.0048

Figure 26. Example set of metrics between a treatment (T) and control (C). A full breakdown is shown, indicating and explaining the overall movement.

Another important aspect of scorecard organization is providing breakdowns of key metrics. For example, Figure 26 shows that a success metric Overall Page Clicks Rate has moved negatively in the treatment group. Such movement is much easier to interpret using the breakdown shown below, which shows that the negative movement happened in “Web Results”, and that its other components were in fact positive.

4) VERIFYING DATA QUALITY

Data quality checks are the first step in analyzing experiment results [24]. One of the most effective data quality checks for experiment analysis is the Sample Ratio Mismatch (SRM) test, which utilizes the Chi-Squared Test to compare the ratio of the observed user counts in the variants against the configured ratio. When an SRM is detected, the results are deemed invalid and the experimenter is blocked from viewing the results to prevent incorrect conclusions.

Another useful data quality verification mechanism is running A/A experiments. In an A/A experiment there is no treatment effect, so if all the computations are correct, the p-values are expected to be distributed uniformly. If p-values are not uniform, it indicates an issue, for example an incorrect variance computation.

Given that data quality issues put in question the validity of OEC and other key experiment metrics, the platform needs to guarantee that these metrics are easily discoverable on the scorecard, and experiment owners are alerted of any degradations.

5) ALERTING

In the early stages of growing experimentation in a product, when only a few experiments are run, alerting is an optional feature of the platform as every experiment is carefully designed and monitored by the experiment owner. As experimentation scales, however, there is less scrutiny about every proposed experiment, and less monitoring. Experimentation platform needs to become more intelligent, automatically analyzing experiment results and alerting experiment owners, providing an important layer of live-site security.

Experimentation portal needs to provide functionality to configure alerts for every product running experiments. Typically alerts would be configured for data quality metrics, OEC metrics, performance metrics such as page load time, errors and crashes, and a few other important product characteristics.

To make alerts actionable, statistical analysis needs to be done to determine when to trigger an alert. Note that the goal of analysis

here is different from that of understanding experiment results – alerts should only notify experiment owners of serious issues that may require experiment shutdown. It is important that alerts take into account not only the level of statistical significance, but also the effect size. For example, an experimentation platform should not alert on a 1-millisecond degradation to page load time metric, even if the confidence that the degradation is a real effect is high (e.g., the p-value is less than $1e-10$.)

Based on the severity of the impact, the system can simply notify the experiment owner, schedule experiment termination and let experiment owner cancel it, or in most severe cases it may terminate the experiment. These configurations need to balance live-site security with allowing experiment owners to be the final authority on shutting down an experiment.

6) DEEP-DIVE ANALYSIS

While the overall scorecard provides the average treatment effect on a set of metrics, experiment owners should also have an ability to better understand metric movements. Experiment scorecard should provide an option to investigate the data by population segment (e.g. browser type) or time-period (e.g. daily/weekly) and evaluate the impact on each segment. Using this breakdown, experiment owners might discover heterogeneous effects in various segments (e.g. an old browser version might be causing a high number of errors) and detect novelty effects (e.g. the treatment group that received a new feature engages with it in the first day and stops using it after that). This type of analysis, however, needs to be done with care, as it is subject to the multiple comparisons issue. Automatically and reliably identifying such heterogeneous movements is an active research area. ExP platform employs one such method [223], notifying experiment owners if interesting or unusual movements are detected in any segment.

To deeper understand the metric change in a specific segment, the system should provide tools to easily obtain examples of user sessions, containing the events that are used to compute the metric

of interest. For example, in an increase in JavaScript errors is observed on a Bing search results page, it may be useful for experimenter to see examples of the error types and queries they triggered on, and conduct simple analysis to see if a specific error type, or a specific query, is causing the increase. While such analyses are conceptually simple, implementing tools and automation to support them is important. Doing such deep-dive analyses manually requires deep understanding of how the data is structured, how a specific metric is computed, and how experiment information is stored in the data, which is beyond the level of knowledge of a typical experiment owner.

9.4 CONCLUSION

Online controlled experiments are becoming the standard operating procedure in data-driven companies [11], [57]. However, for these companies to experiment with high velocity and trust in experiment results, having an experimentation platform is critical [9]. In this chapter, we described the architecture of a large scale online experimentation platform. Such platform is used by thousands of users at Microsoft, supporting hundreds of concurrent experiments, totally over 10000 experiments per year. This platform enables trustworthy experimentation at scale, adding hundreds of millions of dollars of additional yearly revenue for Microsoft. We hope that the architectural guidance provided in this chapter will help companies implement their experimentation systems in an easier, more cost-effective way, helping to further grow the experimentation practices in software industry.

10 EFFECTIVE EXPERIMENT ANALYSIS

Abstract

Online Controlled Experiments (OCEs) are the norm in data-driven software companies because of the benefits they provide for building and deploying software. Product teams experiment to accurately learn whether the changes that they do to their products (e.g. adding new features) cause any impact (e.g. customers use them more frequently). To make informed decisions in product development, experiment analysis needs to be granular with a large number of metrics over heterogeneous devices and audiences. Discovering experiment insights by hand, however, can be cumbersome. In this chapter, and based on case study research at case company H, we (1) describe the standard process of experiment analysis, (2) introduce an artifact to improve the effectiveness and comprehensiveness of this process, and (3) evaluate our work on 86 experiment analysis instances. We demonstrate that with our main contribution which guides the experimenters to the most critical results, companies can accelerate the analysis of their experiments and make the decision-making in software product development more accurate.

This Chapter is based on

A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch, “Effective Online Experiment Analysis at Large Scale,” in submission, shorter version of this paper to **appear in Proceedings**¹ of the 44th Euromicro Conference on Software Engineering and Advanced Applications, SEAA'18, (Prague, Czechia, August 2018).

¹ © 2018 IEEE. With permission from the authors.

10.1 Introduction

Companies are successful only if they truly understand their customers' needs, and develop products or services that fulfil them [6], [28]–[31]. Accurately learning about customer needs, however, is challenging and has always been an important part of software product development. Although asking customers through interviews, focus groups or other similar approaches remains essential for product development, product usage data such as telemetry or product logs [21], [23] enable software companies to become more accurate in evaluating whether their ideas add value to customers [29], [34], [55]. More than ever, companies today realize the importance of learning from data that reveal how their customers actually use the products that they develop [23], [133], [175], [198], on top of the data that reveals what customers say they do with them [59], [64], [215], [216]. One of the enablers of this evolution is the internet connectivity of software products, which provides an unprecedented opportunity to evaluate ideas with customers in near real-time, and creates potential for making fast and accurate causal conclusions between the changes made to the product and the customers' reactions on them [14], [18], [130], [136]. One way in which causal conclusions (for example, introducing a new feature causes customers to use the product more frequently) can be made is through A/B tests, or more generally, Online Controlled Experiments (OCEs) [11], [29], [130]. Experiments reduce the risk from deploying software by minimizing the magnitude and duration of harm caused by software bugs, set directions and goals for product teams, help save infrastructure resources, and deliver many other benefits to data-driven companies [11], [25].

To efficiently and autonomously conduct many OCEs without costly data-scientist involvement, companies require solutions and processes in place to support experimentation [9]. To make informed decisions in product development, individual experiments need to be carefully designed, correctly executed and most importantly, thoroughly analysed. In this work, we focus on the analysis part of OCEs. As researchers in [223] have discovered, effects

of product changes can be heterogeneous across different segments (e.g. an old browser version might be causing a high number of errors, impacting the overall results. Manually examining a large number of segments, however, is time consuming and, based on our experience, often is not exhaustive, missing important insights.

In this chapter, and based on a case study at Microsoft where over ten thousand experiments per year are conducted, we (1) present their current process and approach for analyzing OCEs, and (2) develop an artifact that improves the velocity and comprehensiveness of this process. We validated our artifact with 20 employees at the case company on 86 experiment analysis instances. Our findings indicate that our approach accelerates the analysis of experiments, and improves its comprehensiveness. This can increase the accuracy of deployment decisions in software product development, leading to better products and services.

10.2 Background

Experimentation in software product development is an increasingly active research area [10], [14], [28], [29], [31], [55], [58], [144]. Prior research has been focusing on topics such as new statistical methods to improve metric sensitivity [188], metric design and interpretation [44], [122], projections of results from a short-term experiment to the long term [15], [184], the benefits of experimentation at scale [11], [25], experiments in social networks [37], and rules of thumb and lessons learned from running controlled experiments in practical settings [154], [192]. In the subsections below, we briefly discuss the most relevant parts of prior work on experimentation to provide the context for this research.

10.2.1 Online Controlled Experiments

The theory of controlled experiments dates to Sir Ronald A. Fisher's experiments at the Rothamsted Agricultural Experimental Station in England during the 1920s [129]. In the simplest controlled experiment, two comparable groups are created by randomly assigning experiment participants in either of them; the control and the treatment. The only thing different between the two groups is a change X. For example, if the two variants are software products, they might have different design solutions or communicate with a different server. If the experiment were designed and executed correctly, the only thing consistently different between the two variants is the change X. External factors such as seasonality, impact of other product changes, competitor moves, etc. are distributed evenly between control and treatment and therefore do not impact the results of the experiment. Hence any difference in metrics between the two groups must be due to the change X (or a random chance, that is ruled out using statistical testing). This design establishes a causal relationship between the change X made to the product and changes in user behavior, measured through metrics.

10.2.2 Experimentation Metrics

For effective experimentation, a product team should identify and formalize a set of metrics that should describe long-term goals of the product. Typically, metrics are constructed by combining log signals (for example, clicks, swipes, dwell times, etc.) written in product telemetry and aggregating them over analysis units of interest (e.g per user, per session, etc.) [9], [44], [122], [183]. In this way, informative metrics such as sessions per user or dropped connections per customer can be created, measured, and impacted through experimentation. While designing metrics, it can be intuitive to measure and consider short-term effects [184]. For example, providing more weight to advertisement metrics increases short-term business value and creates business more profitable momentarily. On the long term, however, showing more advertisement could be dissatisfactory to users and increase churn to a competitor

with less advertisement, leading to lower value of the product. Therefore, companies that conduct OCEs invest into developing descriptive metrics that represent long-term product goals and analyze how they or leading metrics [226] that predict them are affected in experiments [227].

As the number of metrics grows, companies organize their metrics in semantically meaningful groups. Conceptually, the following four groups of metrics have been recognized as useful for analyzing experiments in previous research [9], [16], [44]: *Success Metrics* (the metrics that feature teams should improve), *Guardrail Metrics* (the metrics that are constrained to a band and should not move outside of that band), *Data Quality Metrics* (the metrics that ensure that the experiments will be set-up correctly, and that no quality issues happened during an experiment), and *Debug Metrics* (the drill down into success and guardrail metrics)

10.2.3 Experimental Design

Experimentation is a scientific, evidence-driven process that requires a profound understanding of related statistical fundamentals. The most critical statistics are the *power analysis*, *Sample Ratio Mismatch (SRM) calculation*, and the *statistical difference test* [9], [129]. With the Power Calculation, experimenters identify the minimal number of experimental units (for example, users) that they need in the experiment groups in order to detect the desired difference (also known as an effect size). Power calculation is in detail described in [205]. In the SRM calculation, expected distributions between the experimental groups (for example, 50% for the control group and 50% for the treatment group) are compared with the actual observed distributions (for example, 123456/126543) using e.g. the chi-square statistical test. If the SRM calculation reveals a statistically significant difference (for example, the chi-square test returns a p-value lower than conventional 0.05), experiment results should not be trusted. Both power calculation and SRM test are data-quality checks [9], [29] that help experimenters build confidence in interpreting their experiment results. On the other hand, the actual significance between the two

group is typically a combination of a statistical test (e.g. student t-test) that returns a p-value - the probability to observe the given difference (or a larger one) under the assumption that the two groups are not different), and an absolute or relative delta between the two variations [14], [29], [135].

Despite the work described above, there has been no previous research on improving the effectiveness of the experiment analysis process. To address this, we conducted a case study, which we present in greater detail next.

10.3 Research Method

This case study research [137] builds on an ongoing work with Microsoft. A single OCE at Microsoft can be started in a few minutes, most experiments have hundreds of thousands to millions of users, and the company has extensive automation in place to monitor the trustworthiness of experiments.

10.3.1 Data Collection

The study is based on historical data between 2007-2017, collected and analyzed at the case company. It consists of meeting notes, interviews, observations, documentation, and validation recordings and questionnaires. The first author of this chapter has been collaborating with the case company for over two years. During this time, we collected historical data about experiments conducted in the last ten years, participated and took notes at daily meetings where recent experiment analyses were discussed, interviewed case company participants, and conducted evaluation sessions. The second author of this chapter is employed at the case company for over seven years and worked closely with many product groups to help them conduct OCEs. In Table 1, we present the case company participants from Engineering, Data Science and Management roles that were involved in this research. The participants marked with X in the (I) column were interviewed for 30 minutes, whereas the participants in the (E) column participated in the evaluation sessions.

		<i>Role</i>	<i>(I)</i>	<i>(E)</i>
1	DATA SCIENTISTS	Data Scientist	X	X
2		Data & Applied Scientist		X
3		Software Engineer		X
4		Data Scientist		X
5		Data Scientist		X
6		Data Scientist		X
7		Senior Data Scientist	X	X
8		Senior Data Scientist	X	X
		Principal Data Scientist	X	X
9	MANAGEMENT	Principal Software Engr. Mngr.		X
10		GPM PM Manager	X	X
11		Principal Program Manager	X	X
12		Principal Program Manager	X	X
13		Program Manager		X
14		Senior Software Developer Lead		X
15		Principal Group Engineering Mgr.	X	
16	SOFTWARE ENGR.	Senior Software Engineer	X	X
17		Senior Software Engineer		X
18		Software Engineer		X
19		Senior Software Engineer		X
20		Software Engineer		X
21		Principal Software Engineer	X	X

Figure 27. Case company participants that took part in Interviews (I) and Evaluation sessions (E).

10.3.2 Data Analysis

First, and to discover how the experiment analysis process is conducted at the case company, we analyzed our empirical data using thematic coding approach, and triangulated the notes from experiment analysis observations with historical OCEs data and interview notes. Specifically, we were interested in identifying the most common analysis steps used in this process. Second, and to evaluate our approach to improve the analysis process, we implemented an artifact, and analyzed the empirical data related to its evaluation through both thematic coding and statistical analysis.

10.4 Analyzing Online Controlled Experiments

In this section, we present the analysis of online controlled experiments at our case company. The analysis of online controlled experiments is a process where experimental data that were collected from the customer interactions with the product are investigated for their trustworthiness and data-quality, decision making, and insights discovery. Although some of the analysis is conducted before the experiment starts (e.g. computing the required sample size), most of the analysis happens after the experiment is stopped. In this chapter, we focus on the later, and present how our case company analyses experiments both using the Standard Scorecard Analysis (SSA) and through Guided Scorecard Analysis (GSA).

10.4.1 Standard Scorecard Analysis

At our case company, experiments are commonly analyzed through scorecards. In a typical scorecard, the most relevant information about experiment metric movements is displayed. If we take an experiment with one control and one treatment as an example, the scorecard displays, for every metric, the result of the metric for the control and the treatment (for example, an average of clicks), the difference between the two (sometimes expressed as a percentage of change relative to the control - %delta), and most importantly, the p-value, denoting the level of statistical significance.

To make informed decisions, our case company organizes metrics in a scorecard in a semantically meaningful breakdown. For example, (see Figure 28), understanding why a success metric ‘Overall Clicks Rate’ has negatively reacted in the treatment group is easier to interpret with the additional knowledge that it was caused by a decrease of clicks on ‘Web Results’, while other components of the page had click increases. Furthermore, metrics are also grouped in similar conceptual groups as described in Section 2.

Metric	T	C	Delta (%)	p-value
Overall Click Rate	0.9206	0.9219	-0.14%	8e-11
Web Results	0.5743	0.5800	-0.98%	~0
Answers	0.1913	0.1901	+0.63%	5e-24
Image	0.0262	0.0261	+0.38%	0.1112
Video	0.0280	0.0278	+0.72%	0.0004
News	0.0190	0.0190	+0.10%	0.8244
Related Search	0.0211	0.0207	+1.93%	7e-26
Pagination	0.0226	0.0227	-0.44%	0.0114
Other	0.0518	0.0515	+0.58%	0.0048

Figure 28. Example set of metrics between a treatment (T) and control (C). A full breakdown is shown, explaining the overall movement.

During the analysis process, every experiment is being examined for (1) data quality issues, (2) decision making, and (3) deep-dive insight discovery through segment-scorecards similar to the one on Figure 1. We illustrate this with an extract from a document collected at the case company intended for onboarding and guiding experimenters. We discuss each of the steps of experiment analysis in greater detail next.

10.4.1.1 Data Quality Analysis

“To copy common parlance, ‘hope is not a strategy’ when it comes to ensuring that the resulting analysis is trustworthy and reliable enough to provide firm conclusions of your experiment. Various checks need to be performed on the data and the analysis.”

-- Case Company Documentation

Data quality is critical for trustworthy analysis. Controlled experimentation allows for detection of very small changes in metrics that may not be detectable by monitoring the metric over time in a non-controlled setting. Thus, experiment analysis is very sensitive to data quality issues, even those which may not appear big in aggregate. One of the most effective data quality checks for experiment analysis is the Sample Ratio Mismatch (SRM) test, which utilizes the Chi-Squared Test to compare the ratio of the observed user counts in the variants against the configured ratio. When an SRM is detected, the results are deemed invalid and the experimenter is blocked from viewing the scorecard results (in order to prevent false conclusion making). SRM data quality check is performed at many organizations that conduct OCEs, and is frequently highlighted in the related literature [11], [16], [26], [135], [187], [228].

While the SRM check applies to all types of products, every product or feature also contains data quality metrics tailored for their unique purpose. For example, a scorecard for an online web page may contain metrics such as “number of JavaScript errors” and “page load time”, while a scorecard for a mobile app may contain metrics such as “telemetry loss rate” and “number of app crashes”. These and other similar metrics would notify the experimenter about possible issues caused by their experiments.

“Like all scalable systems, we need to design for failure, and part of designing for failure in experimentation is being able to debug and understand the anomalous phenomena that are observed. This aspect of debugging is not necessarily code debugging; it is also data debugging.”

-- Case Company Documentation

Decision Making

“That deeper understanding should drive subsequent decisions around whether to ship a change, iterate on it, or discard it.”

-- Case Company Documentation

Product teams experiment with their design decisions, relevance modifications, infrastructure changes and other types of features in order to conduct a decision. In the early stages of experimentation maturity [9], [229], experimenters optimize their products on experiment specific metrics. Ideally, as companies mature, scorecards should clearly mark the subset of key metrics that the product organization agreed on being optimized for. Commonly, these are known as OEC (Overall Evaluation Criteria) and extensive research exists on designing key success metrics for decision making [11], [16], [26], [29], [135], [187], [228]. A good OEC consists of success metrics - those that the experimenters are aiming to improve and which lead to the long-term desired outcomes for the company, and guardrails - metrics that express business constraints that the company does not want to degrade. For example, for a web site a success metric can be “number of visits per user”, and a guardrail metric may be “ad revenue” (note that “ad revenue” is not a good success metric since increasing it via, e.g., showing users more ads may lead to users abandoning the site resulting in decrease in revenue over a long term [184]).

The decision-making analysis is concerned with identifying how the success and guardrail metrics that the experimenter expected to be impacted in an experiment *actually* reacted to each treatment. For example, if an experiment changed the appearance of an item on a web page with the intent to increase the engagement of the customers visiting the site, has this in fact resulted in the expected outcome? Previous research shows that only one third of ideas deliver the desired positive impact and that about a third of them are counterproductive [11], [29]. Therefore, to get an initial understanding of an experiment, experimenters analyze individual success and guardrail metrics. Although highlighting the metrics that actually changed in an experiment accelerates the analysis process,

understanding the impact on success and guardrail metrics over dozens to hundreds of segments becomes a challenge.

We illustrate this with a quote from one of the case company representatives below.

“Why do we do this to people? Our minds explode with so many metrics. It's like a mystery, like walking through a maze!”

- Principal Program Manager

10.4.1.2 Deep-Dive Insight Discovery

In order to conduct informed decisions, experiments are typically examined beyond the aggregate analysis. At our case company, it is common for experiment scorecards to compute thousands of metrics over dozens of population segments, making manual scorecard analysis tedious and error prone. Important insights can be missed, especially if they are not expected by the experiment owner or heterogeneity is present in the treatment effect [223]. For example, product teams frequently segment their customer base in various ways in order to discover as many insights as possible to base their decisions on. Customers of a web application can be segmented by location, device type, gender, data center that they communicate with, to name a few. In practice, a scorecard similar to the one on Figure 1 needs to be reviewed for every segment, providing experimenters information specific to the segment (e.g. segment size, metric behavior, stat. sig. level).

Segmentation helps in better explaining how customer groups reacted on treatments, and in identifying heterogeneous treatment effects. For example, an experiment treatment can have a large impact on customers accessing the same webpage on mobile in one direction, and on desktop devices in another direction. At scale, such movements are frequent and impact the correctness of decision making process. For example, if a certain segment of customers is more important for the organization from the business perspective, experiments should not degrade the experience of that segment, regardless of how they impact the rest of the customer base. Again, by examining only aggregate data such effects may be

missed. To mitigate this, our case company calculates the statistics for all available segmentations and offers experimenters a detailed scorecard view for both the aggregate data, and for every segment. The result of this is that experimenters need to examine many scorecards to identify insights leading to better decisions. At large scale where many experiments need to be analyzed, or at smaller companies with experimenters that might not have statistical background, this poses a challenge in trustworthiness of decision making.

10.4.2 Guided Experiment Analysis

In the previous section, we described how experimenters examine individual experiments for data quality issues through standard scorecards, that they evaluate the success of their experiment based on success and guardrail metrics, and that they perform this analysis on multiple levels (i.e. for every available segment in an experiment scorecard). For reviewing a typical experiment, tens of different segment slices could be reviewed. This is a cumbersome and error prone approach. In this section, we demonstrate a possibility for a more effective experiment analysis using descriptive visualization for guiding experimenters to the most critical or interesting experiment scorecards, which we name Guided Experiment Analysis (GEA).

In our artifact (a mockup of it is visible on Figure 2), experimenter is presented with a graphical illustration of an experiment at a segment level for every pair of experiment variations. The artifact consists of three columns. The first column depicts data quality behavior of the experiment, the second column contains the OEC behavior, and the third column depicts Segments of Interest (SOI) behavior. The number of rows is dynamic and depends on the available data collected from product telemetry. Every row represents a segmentation criterion which splits the experiment population into multiple segments, for which experimenters at the case company examine individual scorecards. For example, the segment 'Data Center' is divided into slice '1' for the customers connected to the first data center, '2' for those with the second, etc. The

“Gender” segment is also further segmented into “Male”, “Female” and “Unknown” slices, and so on with other segments.

Segment	Data Quality View	OEC View	SOI View
*Aggregate	Aggregate	Aggregate	10
Gender	M F ?	M F ?	2 2 2
Device Type	M W ?	M W ?	1 2 1
Data Center	1 2 3 4 5	1 2 3 4 5	1 2 1 2 8
Location	US	US	10 1

Figure 29. Experiment Analysis Approach.

In the example from Figure 29, analysis is available for all of the product segments, on top of the “Aggregate” segment. An additional data point in this view is the size of the cell, which is proportional to the size of the segment slice. For example, experimenters using our artifact quickly see that the Gender segment slices in this experiment had a similar number of users.

Our artifact operates on sets of metrics. From the experimenter perspective, the artifact reveals whether there has been a significant change or not within a certain slice. At the same time, the experimenter has the ability to examine the slice in greater detail to identify which of the metrics contributed to the decisions by clicking on the individual slice and opening the Standard Scorecard for it. In this way, experimenters are guided from a ‘birds-view’ perspective to the segment slices where the likelihood of finding relevant experiment findings is high. What differs between the columns, and the critical information that leads to an accelerated discovery of critical experimental insights is the way in which the cells are colored.

10.4.2.1 Data Quality View

Experimenters analyze OCEs first for trustworthiness and data quality issues. Therefore, to provide trustworthy experiment analysis, the first column in our artifact highlights data quality issues on a segment slice level. Our artifact colors a cell in the data quality view green when no statistical difference in data quality metrics has been detected. In contrast, when either of the data quality metrics has a statistically significant change above or below a certain threshold (for example, the “telemetry loss rate” increases or decreases), the cell is colored red. In a scenario of a SRM, the cell with its counterpart in the OEC view is disabled and colored black, which can be seen on Figure 1 for the second data center. As this is a critical issue, the experimenters are informed about it also in the OEC view.

10.4.2.2 OEC View

The second column in our artifact summarizes the key success and guardrail metric movement. When no guardrail or success metric movement is present, the cells are colored gray. This informs experimenters that the change in the experiment did not have an impact on that segment slice. When an experiment treatment only positively impacts the success metrics and no negative movement in the guardrail metrics is detected, the segment slice cell is colored green. When there is only negative movement in success or guardrail metrics, the cell color is red. Finally, it is possible for the results to be mixed (e.g. success metrics improve but guardrail metrics degrade), in which case the cell is colored purple, indicating that the tool cannot automatically determine whether this is good or bad. On example from Figure 2, experimenters learn that the “Aggregate” segment is positive, and that it was the male customers that contributed to this, whereas the female in fact reacted negatively. In such scenario, product teams can quickly start to form hypothesis about the possible causes for such heterogeneous movements, and instantly expand the understanding of the impact of their product changes.

10.4.2.3 Segments of Interest View

The third column in our Guided Experiment Analysis is the SOI view. Here, every segment slice is evaluated with an algorithm (developed prior to this research and also available in the SSA), and a score between 0 and 1 that depicts the relevance of its metric movement is assigned to it. Our view colors the cells based on this score. Darker shades of blue indicate segment slices with a high relevance of metric activity. In addition, the cells in this view display a number, indicating the count of metrics that have the activity recognized by the algorithm.

10.4.2.4 Controlling False Discovery Rate

With many metrics being computed in every experiment, the likelihood for a statistically significant change by chance is high. Therefore, we define thresholds to control the statistical significance and the practical significance of metric movements. For statistical significance, we only react to highly significant metric movements (e.g. the p-value < 0.001). For practical significance, we set thresholds on how large the change ($\% \Delta$) for a certain metric needs to be (see e.g. Figure 1). The thresholds are defined by product owners. For example, for “page load time” metric, experiment owners may decide that they only want to highlight the movement of 1% or higher. We tune the threshold experimentally, to provide optimal balance between actual insight triggering and reacting to noise.

10.5 Evaluation & Discussion

In this section, we describe how we evaluated our artifact. We developed our artifact in close collaboration with the study participants listed in Table 1. In the first phase, we interviewed ten participants (see column (I) in Table 1) in order to induce the experiment analysis process, which we described in detail in Section IV.A. During the development, we presented prototypes of our artifact to a wider group of employees at the case company (approximately 45 people) several times. Based on the collected feedback, we developed and integrated our artifact in the case companies' experimentation platform, and evaluated it in detail, which we describe next.

10.5.1 Evaluating Guided Experiment Analysis

We evaluated our artifact with 20 individuals working at the case company. Our goal was to determine whether the GEA approach together with the SSA improves the *velocity* of experiment analysis (the effort spent until a certain insight is discovered) and its *comprehensiveness* (the number of insights and their correctness).

10.5.1.1 Evaluation Study

During our study, we invited participants for an evaluation sessions that lasted 30 minutes. We selected participants from three groups: Data Scientists, Software Engineers, and Managers. Detailed list of participants is visible in Table 1-(E). Experimenters were asked to analyze several OCEs spending about 5 minutes on each. Specifically, we asked them to speak out loud, mentioning the insights that they discover, and to share when their *initial impression of the experiment* (e.g. experiment seems to have a positive impact without hurting any metrics) and their *final hypothesis* (e.g. experiment is positive and it should be deployed to everyone). The evaluation was designed as a controlled experiment; each experiment participants was instructed to use either only the Standard Scorecard Analysis approach, or they could use the Guided Experiment Analysis approach in combination with the SSA approach. We randomly alternated the two options for every participant, giving every participant the ability to conduct an experiment analysis

through both approaches multiple times. The experiments were given to participants with some context, and in an order that prevented learning effects. For example, consecutive experiments differed on, at least, the product type, available segmentation, or types of metric movements that occurred. Every evaluation was recorded, capturing both the activity of the participant and their voice. At the end, the participants were given a questionnaire for feedback on the proposed GEA.

10.5.1.2 Evaluation Metrics

Based on the insights collected during the interviews and group meetings, we agreed on seven metrics as proxies for the two objectives (velocity and comprehensiveness of experiment analysis) . We present them briefly in Table 11.

	<i>Metric Description</i>
TTIH	Time To Initial Hypothesis. The time taken for the participant to form an initial hypothesis.
TTSE	Time To Segment Examination. The time spent until examining at least one segment on top of aggregate.
TTDQ	Time To Data Quality. The time spent until a participant recognizes at least one data quality issue or declares no data quality issues.
TTFH	Time To Final Hypothesis. The time that the participant needed to declare whether to ship the experiment or not.
#DQI	Number of Data Quality Insights. The count of data quality insights that participants discovered.
#OECI	Number of Overall Evaluation Criteria Insights. The count of insights related to the OEC metrics.
#CEI	Number of Critical Insights. The count of other insights that are critical for experiment interpretation.

Table 11. Evaluation Metrics.

10.5.2 Evaluation Results

Analysis Velocity. Through the evaluation studies, we learned that the participants using our artifact in combination with the existing scorecard analysis tools are **37% faster** in providing an initial hypothesis of the experiment outcome, that they are **75% faster** in evaluating at least one individual segment than the aggregated data containing all of the segments. In addition, the participants were 65% faster in declaring the data quality state of the experiment. The time to final hypothesis experiment, however, was nearly **identical**, which is likely due to the time limit (most participants used the full 5 minutes per experiment). To get significant results on TTFH, longer time would have to be allocated to individual analysis instances.

We depict the results of the velocity analysis in Table 12, and illustrate the experience of the participants of the study with example quotes next.

Metric	SSA (s)	GEA(s)	Delta(s)	%Delta	p-value
TTIH	125	79	-46	37%	<0.05
TTSE	205	52	-153	75%	<0.05
TTDQ	203	70	-133	65%	<0.05
TTFH	222	227	+5	-2%	0.8033

Table 12. Analysis Velocity.

“I think that it took me like a minute on the existing tool to figure out which are the segments / metrics that are moving, and in the new tool it was immediately clear.”

- Data Scientist

“I think that the new artifact is a much faster way to see what is going on. I think a lot of people wouldn't look at [all] the segments in the existing tool due to the time constraints.”

- Data Scientist

The quantitative results and qualitative impressions from our participants favor GEA. On top of this, they also reveal another learn-

ing. Many participants using SSA declared their initial hypothesis **before** examining any segments, and even before concluding that the data quality is fine and experiment results can be trusted. This is a **very dangerous practice**. In the GEA the analysis of segments happened before or closer to the initial hypothesis, which is a the correct order of trustworthy experiment analysis.

Analysis Comprehensiveness. In relation to Analysis Comprehensiveness, we identified that the participants using our artifact in combination with the existing tool discover **112% more insights** related to data-quality issues, they discover **113% more insights** related to the Overall Evaluation Criteria of the product, and that they find **double the number** of critical insights. The results suggest that GEA positively and meaningfully impacts the analysis of OCEs. We depict them results in Table 13 below.

Metric	SSA	GEA	Delta	%Delta	p-value
#DQI	25	53	+28	112%	<0.05
#OECI	12	28	+16	133%	<0.05
#CEI	7	21	+14	200%	<0.05

Table 13. Analysis Comprehensiveness.

“The GEA approach tells a much richer story compared to only the standard scorecard.”

- Data Scientist

“This summary tool is much clearer than the Standard Scorecard on its own.”

- Principal Program Manager

“Even though the Aggregate is positive, it is critical for our partners to examine individual segments and make sure that we do not degrade experience for those in the key segments.”

-Senior Data Scientist

After the study, the participants were invited to provide feedback on our GEA approach through a questionnaire (16 participants returned the questionnaire out of 20). From the results that we aggregated, we learned that a large majority of the study participants agree that the proposed artifact improves the effectiveness of the experiment analysis process. In particular, 15 participants agree or strongly agree that the artifact enables the discovery of the insights that could otherwise be missed, and 14 believe that the artifacts accelerates or strongly accelerates the experiment analysis process.

10.6 Threats to validity & Limitations

Construct validity. We conducted this research in a case company with over 12 years of research and development experience in the area of online controlled experimentation. Study participants and facilitators are all very familiar with the design, execution and analysis of OCEs. **External validity.** The main result of our paper is applicable for every company that already conducts controlled experiments and has segment information available for their products. Despite its measurable impact and qualitative confirmations, however, GEA is not as powerful in every context of online experimentation. For example, if there are very few segments available, experimenters can examine them individually. The benefit of our artifact is therefore greatest for product organizations with several informative segments, each containing enough data for statistical validity. Also, and in order to benefit from the proposed artifact, products need to be instrumented and informative metrics such as data-quality and success metrics need to be computed for every segment slice. The highest value of our work can therefore be experienced by product teams conducting large number of experiments or those companies that lack data-scientist capacity and aim to make the analysis interpretation more accessible to other roles (e.g. Software Engineers, Product Managers, etc.).

Evaluation validity. The evaluation study that we designed risks of having participant selection bias, task-selection bias, evaluator bias, participant learning effect bias, and participant response bias.

To improve the evaluation validity, we selected diverse participants to cover different roles, different levels of experience with experimentation, and that work with different products that use experimentation within the company. Experiments were selected by two of the authors based on diversity criteria (different products, different outcomes, etc.). and study participants received exact instructions and help during the study. We controlled for learning effects between experiment analysis and designed the study as an experiment. The participants were not aware of the group that they were in, neither of the measures that we collected. The criterion for every metric increase was defined upfront by the first two authors of this chapter, and the study was piloted on one Data Scientist. Every data point contributing to a metric (e.g. every time a study participant mentioned a certain insight outload) was logged with its time of appearance both during the session itself and triangulated with the recording after the session. We did not control for participant response bias, however, we acknowledge the possibility that no negative feedback on GEA was provided by the experimenters due to not being comfortable of sharing it.

10.7 Conclusion

Interpreting the results of OCEs is challenging even for the most experienced in the field [16], [223]. Experiment insights can easily be missed in the vast sets of segments and metrics, data-quality issues undetected, and the method of online controlled experimentation can quickly become untrustworthy. Efficient experiment analysis is thus critical for successful data-driven product development.

In this chapter, we presented the analysis of online controlled experiments through Standard Scorecard Analysis, and how this can be improved through the proposed Guided Experiment Analysis approach. We evaluated our work and learned that we can accelerate experiment analysis and makes the process more comprehensive. We believe that other software companies that conduct OCEs at scale can achieve similar benefits by incorporating Guided Experiment Analysis. For those that are just starting, however, the Standard Scorecard Analysis might be helpful and informative.

11 THE BENEFITS OF EXPERIMENTATION

Abstract

Online controlled experiments (for example A/B tests) are increasingly being performed to guide product development and accelerate innovation in online software product companies. The benefits of controlled experiments have been shown in many cases with incremental product improvement as the objective. In this chapter, we demonstrate that the value of controlled experimentation at scale extends beyond this recognized scenario. Based on an exhaustive and collaborative case study in a large software-intensive company with highly developed experimentation culture, we inductively derive the benefits of controlled experimentation. The contribution of our chapter is twofold. First, we present a comprehensive list of benefits and illustrate our findings with five case examples of controlled experiments conducted at Microsoft. Second, we provide guidance on how to achieve each of the benefits. With our work, we aim to provide practitioners in the online domain with knowledge on how to use controlled experimentation to maximize the benefits on the portfolio, product and team level.

This Chapter is based on

A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch “The Benefits of Controlled Experimentation at Scale,” **in Proceedings**¹ of the 2017 43rd Euromicro Conference on Software Engineering and Advanced Application, SEAA'17, (Vienna, Austria, August 2017), 18–26.

¹ © 2017 IEEE. With permission from the authors.

11.1 Introduction

Software development organizations and their product development teams are increasingly using customer and product data to support their decisions throughout the product lifecycle [200], [59]. Data-driven companies acquire, process, and leverage data in order to create efficiencies, iterate on and develop new products, and navigate the competitive landscape [200]. This is reflected in the recent software engineering research with many publications on how to change and efficiently conduct controlled experiments (A/B tests) in product management to drive incremental and continuous product improvements [4], [9], [29], [30], [46], [49], [230], [231]. Although there are several ways in which software companies can use customer and product data [59], correctly conducted controlled experiments create a more accurate understanding of what customers value and provide sufficient evidence for concluding inference (for example, drawing a conclusion that a key metric has improved due to a particular change in a product) [29].

The benefit of Controlled Experimentation (CE) on continuous product improvement has been demonstrated a number of times both by research in the academia [49], [59], [230], and in the industry by Microsoft [154], [57], Google [58], Facebook [136] and other software-intensive companies [53], [57]. At Microsoft, and on top of several other benefits, hundreds of millions of dollars of additional revenue annually is generated as a result of controlled experimentation [29]. The impact of experimentation, however, extends beyond the direct revenue gain. CE fundamentally changes the way how the R&D work in product teams is planned, prioritized and measured.

In this Chapter, we provide a case study conducted at Microsoft in which we observed that the benefits of controlled experimentation at scale are in fact threefold. First, on the product portfolio level, CE aids in accurately identifying what constitutes customer and business value, and how the work of product teams contribute to it. Second, CE impacts development on the product level where decisions on product functionality, complexity, quality and infra-

structure needs are made. Third, and what can fundamentally be improved by CE is at the team level where R&D effort is planned, executed and measured. We illustrate our findings with five example experiments conducted at Microsoft. With our contribution, we aim to motivate practitioners in software companies in the online domain to extend their data-driven practices and provide them with guidance on (1) where and (2) how to benefit from CE.

The chapter is organized as follows. In section II we present the background and the motivation for this study. In section III, we describe our research method. We show illustrative examples of our empirical data in section IV. In section V, we present our main contribution – the benefits of controlled experimentation on the portfolio, product and team level. In section VI, we list a few challenges with CE. Finally, we conclude the chapter in section VII.

11.2 Background

Software companies have always been evolving their product development practices. At first, they typically inherit the Agile principles on an individual development team level [12] and expand these practices across the product and other parts of the organization [52]. Next, companies focus on various lean concepts such as eliminating waste [95], removing constraints in the development pipeline [94] and advancing towards continuous integration and continuous deployment of software functionality [53]. Continuous deployment is characterized by a bidirectional channel that enables companies not only to deliver new updates to their customers in order to rapidly prototype with them [31], but also to collect feedback on how these products are used. By evaluating ideas with customers, companies learn about their preferences. In this process, the actual product instrumentation data (for example, features used in a session) has the potential to identify improvements and make the prioritization process of valuable features more accurate [157]. As it focuses on what customers do rather than what they say, it complements other types of feedback data [90] and improves the understanding of the value that the product provides [40], [45]. For example, knowing which features are used in a cer-

tain context helps companies in identifying customer preferences and possible bottlenecks. The intuition of software development companies on customer preferences, can be as much as 90% of the time inaccurate [32], [47], [48]. To identify the 10% and reliably evaluate ideas with customers, CE is becoming the norm in progressive and large software companies [4], [29], [30], [46], [49], [230].

11.2.1 Online Controlled Experimentation

In a standard online controlled experiment (also known as A/B test or randomized trial), users of a software product are randomly distributed between several variants (for example, the two different designs of a product interface A and B) in a persistent manner (a user receives the same experience multiple times). User interactions with the product (e.g. clicks, dwell times, etc.) are instrumented and key metrics (e.g. engagement, task success, etc.) are computed [183]. The differences between the experiment variants (A/B/n) are evaluated on a metrics level to identify which of the versions yields improvements to key metrics that the product and portfolio team strive to impact [29], [56]. In case all the variants are identical, the experiment is known as an A/A test. Standard controlled experiments are typically used in situations where features are added or altered. Special cases of an A/B experiment are experiments where the variants are identical, known as A/A tests, and experiments where instead of adding a new feature to the product, an old feature is removed, known as ‘reverse experiments’. In both cases, and to evaluate the results of the experiments, relevant product metrics are computed. We briefly describe their role in the evaluation of experiments next.

11.2.2 Metrics and Value

The importance of metrics as a mechanism to measure behavior has been recognized and studied in the management domain more than fifty years [232]. Metrics are a powerful tool used in organizations to set team goals, decide which new products and features should be released to customers, and how resources should be allocated [44], [122].

Ideally, a product team should identify and formalize a set of metrics for every product that describes long-term goals. These metrics are commonly labelled as ‘Overall Evaluation Criteria’ (OEC) [176]. An OEC should represent both the *business value* (i.e. a measure of changes that result in an increase of the revenue), and *customer value* (i.e. a measure evaluating customer satisfaction with the product). Dmitriev and Wu [44] have recently demonstrated how to use past “experimentation corpus”, i.e. learnings from past experiments, to evaluate the sensitivity and other relevant parameters of metrics, to help practitioners in selecting suitable OEC metrics.

To identify metrics that are key to the business can be challenging [44], [233]. It can be intuitive to measure and consider short-term effect in OEC metrics [184]. For example, providing more weight to advertisement metrics increases short-term business value and creates business more profitable momentarily. On the long term, however, showing more advertisement may annoy users and increase churn to a competitor with less advertisement, leading to lower customer value. Since the short-term effect is not always predictive of the long-term effect, it should not be the sole component of an OEC [192]. Therefore, and to construct quality metrics, product and portfolio teams in software companies should differentiate between leading and lagging indicators and understanding the mapping between them.

11.2.3 Leading and Lagging Indicators

Motivated by the unpredictability of the market, researchers in the economics disciplines introduced a distinction between two types of indicators [226], [234]. A *‘leading indicator’* is a measurable metric that changes within a short period after a change of conditions occurred, typically following a pattern or a trend. This type of indicators are predictive and lead to a goal that we can directly influence [235]. A *‘lagging indicator’* is an indicator that changes with a delay, typically after several other *‘leading’* indicators have already changed. In economic terms, it is a goal with consequence and value [235]. It consists of a performance gap (e.g. from X to Y) and a time (e.g. by 2020).

Companies should focus in setting product team goals on lead measures and avoid lagging indicators for performance goals. Studies indicate that committing to an achievable goal help improve individual performance and that setting challenging and specific goals can further enhance employee engagement in attaining those goals [236]–[238]. In practice, however, it is very difficult to identify which lead indicators are predictable of which lag indicators [29], [44], [184]. In our work, and as one of the benefits identified in this chapter, we demonstrate that CE enables R&D teams in creating a mapping between *‘leading’* and *‘lagging’* indicators. This has implications for product development. It enables agile teams to focus their work on development activities that improve low-level *‘leading indicators’* which will in turn result in a definitive improvement of a high-level, and typically business value focused, *‘lagging indicator’*.

11.3 Research Method

In this chapter, we report on an inductive case study, which was conducted in collaboration with the Analysis and Experimentation (A&E) team at Microsoft. The study is based on historical data points that were collected over a period of two years and complemented with a series of semi-structured interviews, observations, and meeting participations. In principle, it is an in-depth and single

case study [137], however, our participants are from different organizational units and product teams with fundamentally different product and service offerings. Several of the participants worked in other data-driven organizations before joining A&E team. The A&E team provides a platform and service for running tens of thousands of controlled experiments to customers across almost all major Microsoft products. Its data scientists, engineers and program managers are involved with partner teams and departments across Microsoft daily. The participants involved in this research work are primarily collaborating with Bing, OneNote, Office, MSN.com, Xbox and Skype. The five experiments that we present in section IV are illustrative and representative examples of the work that our participants are involved with at Microsoft.

11.3.1 Data Collection

The data collection for this research was conducted in two streams. First, we collected archival data on past controlled experiments conducted at Microsoft. The first author of this chapter worked with Microsoft Analysis & Experimentation team for a period of 10 weeks. During this time, he collected documents, presentations, meeting minutes and other notes available to Microsoft employees about the controlled experiments, the development of the experimentation platform and organizational developments conducted at Microsoft A&E over the last two years. In total, we collected approximately 130 pages of qualitative data (including several figures and illustrations).

The second stream consisted of three parts. The first author (1) participated on weekly experimentation meetings, (2) attended internal trainings on controlled experimentation and other related topics, and (3) conducted several semi-structured interviews with Microsoft employees. In all the three data collection activities, the first author was accompanied by one of the other three authors (as schedules permitted).

The second author of this chapter has been working with the Analysis & Experimentation team at Microsoft for a period of six years and he has extensive experience with controlled experimenta-

tion. He was the main contact person for the other three researchers throughout the data collection and analysis period and advised the diverse selection of data scientists, managers and software engineers that we interviewed. In total, we conducted 14 semi-structured interviews (1 woman, 13 men) using a questionnaire with 11 open-ended questions. The participants that work with different product teams were invited for a half an hour interview. The interview format started with an introduction and a short explanation of the research being conducted. Participants were then asked on their experience with CE, benefits, pitfalls, and the pain points that they experience while conducting controlled experiments.

11.3.2 Data Analysis

To obtain our learnings, we first organized data that belongs to individual experiments together. In total, we analyzed and grouped the data that relates to over 300 controlled experiments. These data were both qualitative descriptions of individual experiments as well as quantitative data related to individual metrics or metric groups. Next, we read the information about every experiment word by word and extracted commonalities and the differences between the experiments, focusing on the experiment learnings, expected outcomes and unexpected benefits. By extracting these, we constructed codes and emerged with individual benefits. This technique is similar to thematic coding [239] and can be used to identify a theory where existing knowledge is limited. All of our interpretations were continuously discussed with the A&E team to clarify any misunderstandings.

11.3.3 Validity Considerations

To improve the study's construct validity, we complemented the archival data collection activities with individual semi-structured interviews, meetings and trainings. This enabled us to ask clarifying questions, prevent misinterpretations, and study the phenomena from different angles. Meeting minutes and interview transcriptions were independently assessed by three researchers. The first author conducted this research while collaborating with the second author who is permanently employed at the case company as a Principal Data Scientist. This set-up enabled continuous access and insight with the relative stakeholders. However, and since this approach differs from a traditional case study [137], the contributions of this chapter risk of being biased from this extensive inside view. Considering *external validity*, our study involved sub-cases that focus their work on products close to the online domain. And although Microsoft is one company, we emphasize that our sub-cases (for example Bing, Skype, Office etc.) within the longitudinal study are diverse and could itself be considered their cases. Each of them has from several hundred to several thousands of employees. At the same time, several of the engineers, managers and data scientists that provided feedback on our work have experience with CE from other large software companies (specifically from Amazon, Google and Facebook). Taking this into consideration, we believe that the results of our study can apply to other large and software-intense companies that adopt the controlled experimentation methodology at scale.

In the next section, we present five illustrative examples of controlled experiments from our collection of empirical data.

11.4 Empirical Data and Findings

Microsoft conducts tens of thousands of controlled experiments every year across dozens of products. In this section, we present five of these experiments in greater detail to give a flavor of the wide variety of experiments and their benefits. The space limitations and proprietary concerns make it difficult to show all the depth and breadth of our empirical data. Due to these limitations, we select the illustrative experiments that are diverse and yield several benefits, and present them in detail below.

The first experiment with the Office Contextual Bar highlights the use of CE for discovery of instrumentation issues and the need for focusing the experimentation on leading metrics to identify the exact amount of impact on the value of the product. The second experiment illustrates how CE ensures software quality. Next, we show an example experiment with a birthday animation where CE enabled the product team to increase the customer and business value of Bing search page. Next, we present a design experiment with the Bing website where the effect of changing a background color was identified as negative. We conclude the section with an experiment from Skype where CE predicted infrastructure capacity.

11.4.1 Office Contextual Command Bar Experiment

Description: Microsoft Office for mobile is a well-known suite of products designed to increase work productivity. The product team responsible for edit interface in Office mobile apps conducted a design experiment with their Word mobile app. They believed that introducing a Contextual Command Bar (see Figure 30) would increase the engagement (e.g. the number of times the users of the app edit their documents).

Hypothesis: The hypothesis was that mobile phone users will do more editing on the phone because the contextual command bar will improve editing efficiency and will result in increased commonality and frequency of edits and 2-week retention.

Outcome: During the set-up of the experiment, the product team experienced issues with measuring the number of edits. After a few

rounds of troubleshooting, they identified that the telemetry implementation did not accurately log the edit events in the instrumentation pipeline, which resulted in inaccurate interpretation of the initial results. They resolved the instrumentation issues and confirmed this with an A/A test (exposing the control and the treatment group with the same version of the product). The results of a following two-week controlled experiment indicated a substantial increase in engagement (counts of edits), but no statistically significant change in 2-week retention metric.

Learning: The experiment provided the team with two key learnings: (1) Proper instrumentation of existing features and the new feature is essential for computing experiment metrics, and (2) defining metrics that are good leading indicators and that they can expect to see a change in a reasonable timeframe is important.

Benefits: Because of this experiment, the product team (1) resolved an instrumentation issue and (2) identified the exact impact

Figure 30. Office Contextual Bar Experiment.

of introducing this type of design change to one of their products on a leading metric. These learnings were strong evidence to generalize the knowledge and prioritize a similar change to the other products in their portfolio.

11.4.2 OneNote Publishing Pane Experiment

Description: Microsoft OneNote is an information gathering and multi-user collaboration application. In one of the experiments with this application, the product team experimented with a modi-

fication to the ‘Share Pane’. **Hypothesis:** The OneNote team suspected many people wanted to share only a page of a notebook, and that the new feature and user interface would increase overall sharing. With the new experimental feature, the user is given the option to publish a page, rather than the entire notebook (see Figure 32).

Outcome: The controlled experiment indicated positive results on the key leading metrics. The fraction of users that succeeded in progressing throughout the sharing funnel—defined as creating a share link or successfully sending a share e-mail – increased. Users in the treatment group also created fewer new notebooks. This suggested that some users that previously wanted to share a subset of a notebook’s content were going through the trouble of creating a brand-new notebook with that subset of content. The new feature clearly frees these users from going through that trouble. While overall the results were positive, the team saw a mismatch in the number of users in control and treatment for a small subgroup of users using a specific older OS version.

Learning: By examining the experiment in detail, the team learned that instrumentation data was lost when the application crashed during the experiment on the older OS version. This caused unusual movements in metrics and the user mismatch. By investigating this experimental alert, the team discovered that a software bug in the new feature caused a crash in the code path related to the new feature.

Benefits: With this experiment, the product team obtained an important learning. CE can help detect critical bugs that only appear with heavy usage in production, and that experimentation can help validate that a new feature or user interface helps customers. Although products under development traverse rigorous testing steps, many issues become visible only in a production environment, and controlled experiments are a great way to catch them before they hit all users of the product. As a benefit of detecting and mitigating such issues, product teams at Microsoft ensure better quality of their software.

Figure 32. OneNote Share Pane experiment.

Figure 31. Bing Search Results Birthday Greeting Experiment.

11.4.3 Bing Search Results Birthday Greeting Experiment

Description: In an experiment on Bing results page, a product team evaluated the impact of a feature that displays an animated birthday greeting for registered users on the day of their birthday anniversary (see Figure 31).

Hypothesis: With this experiment, the team aimed to promote the feature on Bing search result page to users whom Bing knows they have birthday on the current day. The Bing team hypothesized that this feature would increase usage of Bing and will not introduce any harm.

Outcome: After analyzing the results of the experiment, the team learned that several percent of users who saw the promotion, engaged with it. Since this was a good engagement rate, they decided to deploy the features. The analysis of the offline data, however, showed that there was an abnormal distribution of birthdays on the first of January, likely most of which are not true birthdays.

Learning: The product team considered not showing the birthday experience for users with birthday anniversaries on the first of January. However, the solution they selected was to show it anyway, but increase the prominence of a link stating “Not your birthday?”. This link directs the users on a page to correct their birthday. The results of a follow-up experiment indicated that about 1/5 of users who receive the birthday wish on the first of January initiate the update of their birthday. **Benefits:** The team (1) improved their product and (2) increased its customer and business value. And even though the experiment didn't yield the exact expected result, valuable learnings were obtained leading to a new improved feature eventually deployed. We illustrate this with a quote:

“This is a general pattern with experimentation; finding a surprise that is still valuable.”

– Principal Data Scientist

11.4.4 Bing Background Color Experiments

Description: Bing team conducted controlled experiments with two different background color variations: light grey and dark grey, and compared them to the regular white background. Their goal was to identify the effect of changing the background color on the key metrics.

Hypothesis: Bing team hypothesized that experiments of this type will be successful and that the team should further invest into optimizing the background color of the product.

Outcome: The experiments indicated a strong degradation in key metrics for both variants.

Learning: In this experiment, the product team learned that (1) product changes of this type should not be deployed to the overall user base, and (2) further investment into optimizing background colors should not be prioritized.

Benefits: The teams benefited from this by identifying that (1) changing background color in relation to the regular white version yields negative impact on the key business value metrics, and that they should (2) prioritize other design ideas in the future. In general, there are many possible user interface treatments that could be attempted, nearly infinite in fact, and this method of deciding which to pursue (and avoiding ones that might cause harm) is valuable across multiple products in the portfolio.

11.4.5 Skype Infrastructure Migration

Description: Skype is a well-known product for video and audio communication. Recently, the team completed migration of the product to a new call-routing algorithm.

Hypothesis: Among other effects, the team hypothesized that enough infrastructure capacity will be available to support the new approach to call routing.

Outcome: To evaluate their hypothesis, the Skype team conducted a controlled experiment with only a few percent of Skype users. The difference in load on the infrastructure between the new and the old algorithm was estimated, and projected to 100% of user population, confirming that the capacity will be sufficient.

Learning: The Skype team learned that based on the effect of the limited treatment group, enough capacity is available within the new infrastructure to continue with the transition.

Benefits: By identifying the exact effect on the infrastructure, the team was confident that the infrastructure capacity is within the range needed for the complete product migration.

11.5 The Benefits of Controlled Experimentation

In the previous section, we presented five illustrative examples of controlled experiments conducted at Microsoft. In this section, and based on the analysis of several hundreds of experiments, interview and workshop notes, we provide (1) a comprehensive list of benefits of controlled experimentation, and (2) guidelines on how to achieve them. We discuss these benefits in relation to example experiments from previous section IV.

In our work, we differentiate between three distinct levels of benefits. First, we describe how CE benefit the company portfolio level, which stretches across the organization and typically across several products. Second, we describe the benefits of CE on the product level. There, we show how CE controls product complexity, quality, instrumentation assurance, and how it can be used to predict infrastructure needs. Finally, and on the team level, we show that controlled experimentation impacts the planning of team activities and how it can be used to define measurable and realistic performance goals for teams. As illustrated with our examples from section IV, a single experiment can yield more than one benefit. We present each of the identified benefits and how to achieve them in the remainder of this section.

11.5.1 Portfolio Level

Value discovery and validation: Effective portfolio development and identification of what delivers value to customers is vital to successful product innovation [29], [56]. By conducting a controlled experiment, software companies can measure the exact amount of impact on the value from different features and prod-

ucts. In this context, controlled experimentation forces product development teams across multiple products in the portfolio to (1) define what the value is, and to (2) identify what changes across the product portfolio contribute to it. By conducting many controlled experiments and analyzing their outcomes, companies can learn what is valuable for the business, and what is valuable for the customers. The procedure to discover and define business and customer value is the following: (1) The portfolio team hypothesizes customer or/and business value, (2) value measurements are formalized in terms of leading metrics, (3) hypotheses are evaluated in multiple experiments across multiple products, and finally, (4) hypotheses that were accepted define customer or/and business value. Step (3) is crucial and typically consists of several learning experiments (e.g. intentional degradations) where leading metrics that indicated changes to lagging metrics in (2) are discovered (e.g. the effect of a performance slowdown on user satisfaction).

The Office Contextual Bar Experiment and the Birthday Greeting Experiment that we presented in section IV, for example, provided the Microsoft with knowledge of portfolio benefits about what customers of these products value. The learnings were validated and applied across the Microsoft portfolio. A team of data scientists is continuously working on improving the leading metrics, by analyzing past experiments and conducting new learning experiments to gain insights into metrics behavior and learn the mappings between ‘lagging’ and ‘leading’ indicators. Without CE, changes to metrics and correlations between metric movements are much harder to detect due to large variance, seasonality effects, and other factors, making establishing a clear relationship between ‘leading’ and ‘lagging’ indicators nearly impossible.

11.5.2 Product Level

Incremental product improvements: This is the most well-known benefit of A/B testing in software product development and there are several publications available that discuss it in detail [56], [58], [154], [192]. Causality with regular and incremental product improvements (e.g. what was the effect of introducing a new feature on the key metrics) is proved by running a controlled experiment. And by deploying the changes that increase the key metrics, the product is incrementally improved. The way how incremental product improvements are done is the following: First, (1) instrumentation data (e.g. clicks per user session, dwell time, etc.) are collected during an experiment, (2) key metrics are calculated based on the instrumentation data, (3) Statistical tests are conducted to detect the difference between the variants (e.g. the version of the product without a new feature and the one with the new feature) is measured. Finally, (4) variants with improvements to the key metrics are deployed. This benefit is illustrated in several experiments in section IV.

Optimizing and predicting product infrastructure and capacity needs: When a product is incrementally updated (e.g. an algorithm is optimized) companies risk introducing an effect on the infrastructure. By using controlled experimentation, practitioners can be more proactive and guarantee that (1) necessary capacity will be available for their changes, and (2) estimate the capacity cost/savings of the change before it is fully deployed. This can be achieved by measuring the change in infrastructure load via an experiment on a small population, e.g. 10% vs 10%, and then project what will happen at 100% deployment. A gradual ramp-up of the experiment can be used to estimate non-linear effects.

Skype infrastructure migration experiment described in section IV, for example, enabled the product team to predict infrastructure needs based on a smaller treatment and continue the deployment.

Ensuring product quality: Although the changes to products are typically well tested using end-to-end and unit tests, certain scenarios in product changes require validation with production custom-

ers. By experimenting with customers in a gradual manner (e.g. starting with a low % in the treatment group and increasing exposure over time), the impact of a change that introduces product issues is limited to only a defined number of customers. As a side benefit, the investments into virtual testing infrastructure mimicking production environment are minimized. The risk of exposing product changes with defects that could not have been detected through unit and pre-production testing to large sets of customers are lower with controlled experimentation than they would be without it. OneNote Share Pane experiment, for example, provided the evidence that product quality can be better assured using CE by, for example, alerting for instrumentation discrepancies during experiment execution.

Stabilizing and lowering product complexity: Features are developed because teams believe that they will be useful and provide value. Yet in many domains, most ideas fail to improve key metrics [29]. Only one third of the ideas tested on the Experimentation Platform at Microsoft improved the metrics they were designed to improve [29], [154]. CE enables product development teams to (1) deploy only the changes (e.g. new features) that deliver an increase in business or customer value, and (2) to remove those that do not.

The way in which this product benefit is achieved is twofold: (1) features that have no impact on the business or customer value (e.g. do not indicate a statistical significance in key metrics) are not deployed to customers. Second (2) by running reverse experiments (e.g. a feature is hidden from the treatment group), product teams get an indication on how much value the removed feature adds to the product.

Product instrumentation quality assurance: As companies increasingly rely on instrumentation to measure their successes and failures, controlled experiments should regularly be performed to identify whether the metrics are up to date and react as it is expected from them. This benefit can be achieved in two different ways: (1) A/A tests should continuously be conducted (e.g. every time before starting an A/B experiment). Second (2) by conducting experiments with known outcomes, the instrumentation quality

and logic can be verified by comparing the expected and actual outcome. At Microsoft, CE is used regularly to evaluate instrumentation [44].

11.5.3 Team Level

Team activity planning: Experiments provide learnings that consist of treatment and causal effect pairs. These learnings can be used to identify the development activities that should be prioritized (for example, the changes teams made that caused the most impact). By (1) knowing what changes improved the key metrics in the past and by how much, product development teams can (2) generalize and organize their schedule to focus their work on the type of changes that were shown to have the most beneficial effect on the product.

The Successful Office Contextual Bar Experiment enabled sibling teams working on related products to prioritize their work and capture customer and business value earlier than they would otherwise.

On the other hand, the Bing Background Color experiments yield the negative learning and was used to deprioritize similar development activities.

Defining performance goals for teams: With CE, product management can express team goals in terms of changes to ‘leading metrics’, and their progress can be monitored over time. For example, Bing’s ranking team has their goal defined by a single metric, a leading indicator for lagging metrics such as long-term query share and revenue. The team’s progress towards that goal is measured as a sum of the statistically significant changes to this metric from all the controlled experiments performed by the team over a period. Without CE, it is very difficult to accurately evaluate team progress since measuring the exact magnitude of impact from new features is hard, and the mappings between ‘leading’ and ‘lagging’ indicators are unknown.

We summarize the benefits that we described in this section and guidelines to achieve them in Table 14

	Benefits	Guidelines to achieve the benefit
Portfolio	Value discovery and validation	<p>(1) Customer and Business value are hypothesized on a portfolio level.</p> <p>(2) Measurement of the value is formalized in terms of leading metrics.</p> <p>(3) Hypotheses are evaluated in multiple exper. across multiple products.</p> <p>(4) Hypotheses that were confirmed indicate value on a portfolio level.</p>
Product	Incremental product improvements	<p>(1) Instrumentation data of a single experiment are collected,</p> <p>(2) Metrics are calculated based on the collected data,</p> <p>(3) Statistical difference between variants is measured,</p> <p>(4) Variants with improvements to key metrics are deployed.</p>
	Optimizing and predicting product infrastructure and capacity needs	<p>(1) Change is deployed on a low % of treatment,</p> <p>(2) Changes in infrastructure are monitored,</p> <p>(3) Treatment group is gradually increased if resources allow.</p>
	Ensuring product quality	<p>(1) Product changes that degrade key metrics are not deployed.</p>
	Stabilizing and lowering product complexity	<p>(1) Product increments with no impact on the key metrics are not deployed.</p> <p>(2) Product features with no impact on the key metrics are removed with reverse experiments.</p>
	Product Instrumentation quality assurance	<p>(1) A/A experiments are conducted to identify noisy instrumentation.</p> <p>(2) Experiments with known outcomes validate instrumentation quality.</p>
Team	Team activity planning	<p>(1) Changes/features that improve key metrics are shared among teams.</p> <p>(2) Teams generalize learnings to identify feature areas that should be prioritized.</p>
	Defining performance goals for teams	<p>(1) By measuring the exact amount of impact that changes of one team had on the leading metrics over a period, a realistic goal can be set for the next period.</p>

Table 14. Benefits of controlled experimentation and guidelines on how to achieve them.

11.6 Challenges with CE

In the previous section, we demonstrated that the benefits of controlled experimentation at scale extend beyond the recognized scenario of incremental product improvements. To implement CE in a software company and experience the benefits, however, considerable effort and domain knowledge are required. In this section, and to contrast the positive implications of the benefits above, we present and discuss several challenges with CE.

Gradual realization of benefits: Several product teams at Microsoft experience the benefits that we present in Section V in their development process. The realization of them, however, is not instant, but a gradual process of evolvement over time [9]. Companies typically start to conduct controlled experiments on significant changes (for example, a new interface) to gain traction and support for investments into A/B testing infrastructure. As they scale, and as we show in [9], every bug fix and minor change is a priori released through an A/B test. Optimizing the planning of team activities by using CE, for example, can only be achieved at scale [9], [57].

Evolvement of skills and technology: To scale CE, several new skills such as Data Scientists and Data Engineers [18] are needed. At the same time, investments in an experimentation platform and instrumentation of existing products are needed [23]. And although these are initially a cost for the company, a significant learning and ROI are seen over time at Microsoft and other companies [9], [154].

Mitigating pitfalls: Related literature [14]–[16], [187] identified several pitfalls of using CE. As the usage of controlled experiments grows, it is becoming more important to understand the opportunities and pitfalls one might face when using CE in practice. For example, *survival bias detection* in long-lasting experiments (i.e. experiments that last over 10 weeks), *underpowered experiments* (i.e. example experiments with insufficient number of users), *hypothesizing after results are known*, *incorrectly computing confidence*

intervals for percent change, etc. These are pitfalls that make CE less trustworthy and need to be controlled and mitigated.

Prioritization of changes: Experimenting in production environment is becoming the norm for product changes in experimentally evolved online companies that operate in non-safety critical domains [53], [57], [58], [136], [154]. Investments in existing parts of the development pipeline that are resource demanding today (for example in pre-production test systems) are becoming deprioritized due to CE. As a consequence, software companies that evolve CE will increasingly prioritize experimenting in the production. This introduces two types of challenges: (1) deciding which product changes to prioritize for CE vs. regular pre-production testing, and (2) timing the experiments. With CE, it is common to reveal features under development to a large set of users. Companies that study their competitors can exploit this situation in order to earlier predict the direction of the software company.

11.7 Conclusions and Future Work

The benefit of Controlled Experimentation (for example A/B testing) in incremental product development has been demonstrated a number of times both by research in the academia [49], [59], [230], and in the industry by Microsoft [154], [57], Google [58], Facebook [136] and other software-intensive companies [53], [57]. In this chapter, and based on a case study at Microsoft, we demonstrate that benefits of controlled experimentation extend beyond this recognized scenario. First, we show the benefits on the product portfolio level, where CE aids in accurately identifying what constitutes customer and business value, and how much exactly the work of product teams contribute to it. Next, we show that CE impacts development on the product level where decisions on product functionality, complexity, quality and infrastructure needs are made. Finally, we show that CE benefits the team level where R&D effort is planned, executed and measured. We illustrate our findings with five illustrative experiments conducted at Microsoft at several product teams.

Initial guidance on how to achieve certain benefits from continuous experimentation has been presented in the literature both by us [46], [154] and other researchers [4], [10], [28]. In this chapter, we provide guidance also for achieving the benefits related to the portfolio and team level. In our opinion, however, more work is needed in this area. Specifically, we believe that the research and industry would benefit from (1) a deeper understanding of the experiment lifecycle, (2) guidance on detection of patterns between leading and lagging metrics, and (3) providing automation and support for product teams in this process. Also, further studies are needed to identify in what contexts to allow CE, and where not to use this methodology.

12 THE CURRENT STATE OF EXPERIMENTATION MATURITY

Abstract

Online Controlled Experiments (OCEs, also aka A/B tests) are one of the most powerful methods for measuring how much value new features and changes deployed to software products bring to users. Companies like Microsoft, Amazon, and Booking.com report the ability to conduct thousands of OCEs every year. However, the competences of the remainder of the online software industry remain unknown, and the obstacles that stand in their paths obscured. The main objective of this Chapter is to reveal the current state of A/B testing maturity in the software industry, and to discover the main challenges in growing experimentation. We base our findings on 44 responses from an online empirical survey. Our main contribution of this chapter is twofold. First, we reveal the current state of experimentation maturity in the online software industry. Second, we identify the key challenges in growing experimentation as reported by the survey respondents. With our results, we hope to steer future research in the area of Online Controlled Experimentation, and to provide companies that are growing their capabilities a baseline for comparison.

This Chapter is based on

A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch, "Online Controlled Experimentation at Scale: An Empirical Survey on the Current State of A/B Testing," in submission, shorter version of this paper to appear in **Proceedings**¹ of the 44th Euromicro Conference on Software Engineering and Advanced Applications, SEAA'18, (Prague, Czechia, Aug. 2018).

¹ © 2018 IEEE. With permission from the authors.

12.1 Introduction

The internet connectivity of software products provides an unprecedented opportunity to learn what customers value in near real-time, and to make causal conclusions between the changes made to the product and the customers' reactions on them [14], [18], [130], [136]. One way to achieve this in connected products is through A/B testing, or more generally, Online Controlled Experiments (OCEs) [29], [130]. OCEs transform decision making into a scientific, evidence-driven process—rather than an intuitive reaction [11], [29]. And as many large online companies such as Google, Microsoft, and Amazon report on the benefits that they experience due to the growth of the experimentation practices in their companies [25], [47], [56]–[58], the ability of the remainder of the software remains unknown [4], [10], [36], [59]. Conducting a few isolated experiments is relatively easy. However, systematically conducting trustworthy experiments at large-scale (e.g. hundreds or thousands of OCEs per year) requires a transformation that is far from intuitive and easy to achieve. For example, building data pipeline is challenging, and conclusions can be entirely wrong when experiments interfere with each other, or when the randomization of users is biased by previous experiments [15], [16], [187]–[189].

In our previous research, we described the evolutionary stages of experimentation growth in detail through the Experimentation Growth Model (EGM model) [9], [229]. Our experience from applying A/B testing in various Microsoft products for the last 10 years, as well as the experience at other experimentation-driven companies that we collaborate with, has been that enabling and growing experimentation is a gradual process full of challenges and pitfalls [15], [16], [187]. For example, experimentation at scale requires developing strong experimentation practices based on trustworthy statistical analyses, advanced engineering capabilities, setting clear measurable business goals, and alignment of the organization and culture to experimentation. We are confident that due to the growing awareness of the need for trustworthy data-driven

decision making in software product development, and also due to the technical advances and challenges that accompany them (e.g. the complexities brought by AI services), companies will increasingly adopt and evolve their experimentation capabilities in the future. Evaluating ideas with OCEs, as illustrated in prior research [11], [25] yields benefits beyond finding out the better variation of a few software features. If a company grows the experimentation capability and organizational skills to conduct OCEs at large scale, it will be able to assess not only ideas for websites but also potential business models, strategies, products, services, marketing campaigns, etc. [11], [25]. Knowing how well the companies that already experiment are evolved, and what challenges prevent these companies from advancing their experimentation capabilities is therefore of great interest for both researchers and industry practitioners working in this area.

In this chapter, we explore the current state of experimentation maturity based on the EGM model. Our main research question is: **“What is the current state of experimentation maturity in online software companies that already conduct experiments?”**

To answer our research question, we conducted an online empirical survey and analyzed the responses from participating companies. In total, we received 44 valid responses from product team working in companies worldwide. Based on the analyzed data, we present the current state of experimentation maturity, and identify the greatest challenges that prevent companies from advancing.

12.2 Background

In this section, we present the related literature. We start by introducing the need for learning from customer feedback and product data. Next, we introduce experimentation and briefly discuss how companies use this methodology to make trustworthy decisions in software product development. Finally, we introduce the Experimentation Growth Model, on which we base our work in this chapter.

12.2.1 Learning From Data

Companies have always been collecting data to understand what the customers value and make decisions based on the lessons learnt [38]. Although techniques for collecting customer feedback have traditionally focused on the collection of what the customers say they do (for example interviews or focus groups) [182], the software industry evolves towards complementing these with the collection of data that reveals what customers actually do with them [10], [183]. Also, the growing use of Machine Learning, Artificial Intelligence and other algorithms that result in non-deterministic behavior of the software make the use of quantitative feedback on top of the qualitative input even more important. The use of data collected directly from the products enhances the accuracy of evaluating ideas with customers and moves companies away from assumptions and towards trustworthy decision making.

The internet connectivity of software products provides an unprecedented opportunity to evaluate hypotheses in near real-time. Consequently, the number of hypotheses that product management generates and aims to evaluate can be tremendous. For example, there are billions of possibilities merely to style a product (e.g. ‘41 Shades of Blue’ OCE at Google [57]). In fact, style and content management is only one area for experimentation. Evaluating improvements to ranking algorithms and recommender systems [15], [184] are popular applications of online experimentation, among others. And although there are several ways to evaluate hypotheses (with e.g. pattern detection, classification, etc.) none of them shows

as direct causal and accurate impact of the idea on the customer value as Online Controlled Experiments do [184].

12.2.2 Online Controlled Experiments

The theory of controlled experiments dates to Sir Ronald A. Fisher's experiments at the Rothamsted Agricultural Experimental Station in England during the 1920s [129]. In the simplest controlled experiment, two comparable groups are created by randomly assigning experiment participants in either of them; the control and the treatment. The only thing different between the two groups is some change X. For example, if the two variants are software products, they might have different design solutions or communicate with a different server. If the experiment were designed and executed correctly, the only thing consistently different between the two variants is the change X. External factors such as seasonality, impact of other product changes, competitor moves, etc. are distributed evenly between control and treatment and therefore do not impact the results of the experiment. Hence any difference in metrics between the two groups must be due to the change X (or a random chance, that is ruled out using statistical testing). This design establishes a causal relationship between the change X made to the product and changes in user behavior.

12.2.3 The Experimentation Growth Model

In previous research, we inductively derived the Experimentation Growth Model (EGM model) from analyzing the experience of growing experimentation activities in over a dozen Microsoft products and further detailed our work through case studies at Skyscanner, Booking, and Intuit [9], [229].

The EGM model depicts experimentation growth as four stages of evolution, starting from Crawl where experimentation is ad-hoc, time-consuming, and provides limited value, to Walk to Run to Fly, where experimentation is integral part of every aspect of product development, enabling data-driven decisions at scale. ‘Crawl’ is the initial stage where software companies start to experiment, which typically happens within a single feature team and without an experimentation platform. In the ‘Walk’ stage, companies develop the capability to conduct tens of experiments per annum. In the ‘Run’ stage, multiple feature teams conduct hundreds of experiments yearly with an experimentation platform that controls for common pitfalls and alerts experimenters for bad experiments. Finally, in the ‘Fly’ stage, software companies are able to conduct thousands of OCEs yearly, evaluating not only new features, but also every bug fix through experiments for unexpected outcomes.

In the EGM model, the evolution of experimentation from one stage to the next advances along the seven most critical dimensions of experimentation, which we inductively derived in prior research:

- DM1: ‘Technical focus of product development activities’: focuses on the engineering work (for example, building a reliable data pipeline).
- DM2: ‘Experimentation platform capability’: focuses on the features that companies need in their experimentation platform.
- DM3: ‘Experimentation pervasiveness’ assesses the extent to which experimentation is being conducted in software companies (for example, from a few user interface experiments towards experimenting with every change).
- DM4: ‘Feature team self-sufficiency’: assesses the extent to which individual feature team members manage most of the experiments without the involvement.
- DM5: ‘Experimentation team organization’: describes how to organize the experimentation experts in each of the four stages, in order to experience the most benefit from conducting OCEs (for example, from no experimentation team to a center of excellence).
- DM6: The ‘Overall Evaluation Criteria (OEC)’: entails how the key metrics are defined (for example, from a few signals to a structured and well-defined set of metrics that are indicative of long term success of a change which is measured in a short-term experiment).
- DM7: ‘Experimentation Impact’: reveals the extent to which the experimentation impacts the software company at each of the four stages (for example, from an impact to a small feature, towards an impact that sets strategic and team goals).

For a detailed description of the dimensions see [9], [229].

12.3 Research Method

In this section, we describe our research approach.

12.3.1 Experimentation Growth Survey

The first two authors jointly designed the questions for the survey based on the EGM model introduced in Section II.C, and implemented it on the web. The survey [214] asks participants to complete a questionnaire with 14 closed multiple choice questions and 2 open-ended questions. The questions are designed to determine the stage along each of the dimensions of the EGM model. For example, for the Overall Evaluation Criteria (OEC) dimension, the questions “Do you have a formally defined OEC?” and “Do you have a formal process to update the OEC?” are used. The results of the survey are auto-analyzed, and the user is presented with their Experimentation Growth Score, and an evaluation of their progress along each of the dimensions. The detailed breakdown along the dimensions allows the participants to identify those that are evolved, and those that lag behind (e.g. see Figure 1).

We piloted the survey within Microsoft (51 test entries) with product teams at different levels of Experimentation Growth, and with two other companies. In our pilot rounds, the test participants were asked for feedback both on questions and options to answer them, as well as on the mapping of the results. Each of the participants that provided test entries was asked to qualitatively assess their products’ maturity with respect to experimentation prior to the automatic analysis of the results. During the design of the survey, we were constantly comparing the ground truth of the product experimentation maturity (which we had available for 12 product teams within Microsoft, and 2 other companies), and the results of our application.

12.3.2 Data Collection

We published the survey at <https://www.exp-growth.com/>. The target audience for our survey were experimentation owners and experts within software organizations. To better reach the target audience, the first two authors advertised the survey through contacts, on LinkedIn, and physically at two conferences (KDD17 in Halifax, Nova Scotia, and SIGIR17 in Tokyo, Japan). At both conferences, the second author together with the Analysis & Experimentation team from Microsoft hosted a tutorial on Online Experimentation. This enabled us to further connect with the target audience, and ask companies that conduct OCEs for feedback, and to fill-out the survey.

In total, we received 44 valid responses (after removal of 5 duplicates). For consistency, we will use the term “product organizations” to refer to each of the 44 individual entries as we captured multiple responses from one company (large companies have multiple business). The data were collected anonymously and cannot be tied to individual respondents.

12.3.3 Data Analysis

Collected data is both of quantitative and qualitative nature and it was analyzed by the first two authors of this chapter in two steps. First, for reproducibility and real-time feedback to survey respondents, we programmed JavaScript code that computes the score for each dimension of the EGM model based on the responses and assigns it a score of 1 to 4 (crawl=1, fly=4). The final score (EG score) is then calculated by taking the scores along each dimension, adding a score for “scale” (the number of experiments run in a month) and a score for agility (how often the product deploys), and then normalizing the result to produce a value between 1-10. Visually, the values between [0,4) map to the ‘Crawl’ stage, values between [4, 6) map to ‘Walk’, the values between [6, 8) map to ‘Run’, and finally the values between [8, 10] map to the ‘Fly’ stage. The EG score is an indicator of the capacity to run many experiments in a trustworthy manner, and to derive value from their results.

In the next step, we analyzed these computations together with the raw data using descriptive statistics (e.g. counting the number of features reported in the companies' experimentation platform). In addition, qualitative data collected in two of the questions in our survey were analyzed using thematic coding approach in which we grouped similar concepts together.

12.3.4 Threats To Validity

Construct validity. We conducted this research as part of an ongoing work with Microsoft's Analysis and Experimentation team, which has over 12 years of experience in the area of online experimentation. Their input, as well as the feedback we received from other companies, improved the accuracy of the EGM model and the survey study. To assure the respondents familiarity with our work, the EGM model was presented to the survey participants (at the tutorials and on the web) first. They were asked to complete the survey next. We also introduced analytics and data-quality checks in the survey to prevent bots from submitting results and to detect duplicate entries.

External validity. Our survey response solicitation was targeted to the participants from companies that conduct OCEs, and due to the nature of its collection suffers from *selection bias*. For example, we only captured the responses from companies that already run at least a few OCEs per year. Also, participants may have provided idealized data about their experimentation maturity, so we risk of having a *self-reporting bias*. For example, the respondents could assign a higher or lower ranking. Generalizing our results is therefore a limitation that we acknowledge, and future follow-up studies will be needed on larger samples. However, since we captured the experience of companies across the EGM model, i.e. from those that are less mature, as well as from the companies that are very well evolved, the findings in the chapter could be useful for a large number of companies to compare their experimentation capabilities to the remainder of the industry.

12.4 Survey Findings & Implications

In the previous sections, we introduced the dimensions of the EGM model along with other related works on experimentation, and described our data collection and analysis steps. In this section, we present the results of our online empirical survey. We start by first presenting the product organizations that responded to our survey, and we reveal their aggregate results. Next, we discuss the results for each of the seven dimensions of the EGM model individually. Finally, we conclude the section by summarizing our key findings.

Note that the results in this discussion will be reported per product organization and that this should be taken into consideration when interpreting them.

12.4.1 Product Organizations

In our survey, we collected several data points that describe our

sample. Specifically, we asked the respondents for their Monthly Active Users (MAU) count, and their Deployment Frequency. Nine

product organizations reported less than 10,000 MAU, three reported between 100,000 and 1,000,000 MAU, and 32 product organizations report over 1 million Monthly Active Users. Four out of 44 product organizations need longer than a month to ship new code to customers, seven are able to deploy new code monthly, twenty-one deploy new updates weekly, and finally 12 organizations have an ability to ship new versions of their product daily or more frequently. These numbers show that, while our sample is small, it still captures quite a diverse range of product organizations.

12.4.2 Experimentation Maturity

The first step in our analysis is presenting the aggregate maturity results, which we summarize in Table 15. Our analysis reveals that the majority of product organizations (24 out of 44) classify within the Run stage – implying that the majority of them succeeded in implementing data-collection practices with a reliable data-pipeline, enabling multiple feature teams to experiment on most new features added to the product. The aggregate data also reveals that at least five product organizations that responded to our survey succeeded in fully evolving their experimentation capabilities, enabling most individuals in product teams to autonomously conduct experiments with little or no support from the data scientist.

Exp. Maturity Stage	Crawl	Walk	Run	Fly
#Product Organizations	3	12	24	5

Table 15. Aggregate Experimentation Maturity.

On the other hand, twelve companies classify within the “Walk” stage, and only 3 companies classifies under “Crawl”. To provide more insights, we discuss these results by drilling-down to each of the seven dimensions illustrated on Figure 34.

In the remainder of this section, and for every dimension of ExP growth, we (1) briefly repeat its role, (2) present the empirical findings, and (3) prescribe the possibilities for companies to grow their experimentation capabilities.

Figure 34. Experimentation Maturity Breakdown for every dimension of the Experimentation Growth Model.

12.4.2.1 DM1: *Technical Focus of product development activities.*

The first dimension of the EGM model focuses on the engineering activities related to experimentation (e.g. coding the instrumentation of a product, building a data collection pipeline, computing metrics, and standardizing the engineering processes). Based on our survey, **the large majority of product organizations classify in the ‘Walk stage’**. To arrive to this stage, companies standardized data collection and processing practices, and created standard sets of simple metrics. In our previous research [16] we learned that conceptually, the following four groups of metrics are useful for analyzing experiments: Success Metrics (the metrics that feature teams should improve), Guardrail Metrics (the metrics that are constrained to a band and should not move outside of that band), Data Quality Metrics (the metrics that ensure that the experiments will be set-up correctly, and that no quality issues happened during an experiment), and Debug Metrics (the drill down into success and guardrail metrics). By grouping metrics together, deciding on experiment success can be done more accurately and effectively. For example, if data quality issues arise in an experiment, typically multiple metrics will respond and reflect on them. Examining these metrics together provides a quick insight into underlying change.

In order to advance along this dimension, investments in automated reliable data pipelines that join data from multiple teams, designing a detailed breakdown of metrics, and experiment analysis automation are needed from product teams. One example of how companies can advance along this dimension is to construct a detailed breakdown of metrics in a scorecard. The most critical information about an experiment (the metric result in the treatment and the control group, the difference between the two, and whether the difference is statistically significant) should be presented.

12.4.2.2 DM2: Experimentation Platform Capability.

This dimension is concerned with the maturity of the experimentation platform (e.g. to what extent it automates various tasks and report generations). With respect to this dimension only one product organization relies on manual data-collection and coding of experiments. Therefore, the majority of the product organizations use an experimentation platform which enables them to conduct trustworthy randomization of their customers and reliable computation of results. In Table 16, we present the distribution between the types of platform that the companies use, revealing that the majority of the product organizations uses an in-house developed platform for running experiments.

Type of Experimentation Platform	#PO
No platform (manual coding of experiments)	1
Third party platform	11
Internally developed platform	32

Table 16. Experimentation Platforms.

The capabilities of the experimentation platforms between product organizations differ significantly. Specifically, platforms support various functionality, enabling different levels of automation and trustworthiness. For example, to move from the ‘Run’ stage to the ‘Fly’ stage, the experimentation platform needs to support advanced features such as alerting on bad experiments, control of carry-over effects, experimentation iteration, etc., which only 2 out of 44 product organizations succeeded in implementing. To further understand the maturity of the experimentation platforms in the online industry, we analyze their common features (based on [229]), and provide frequency count of their integrations in Table 17.

ExP Feature	Feature Description	#PO
Randomization	Assigning users/devices to variants	37
Targeting	Defining the audience for experiment	36
Scorecard generation	Automatically generates scorecards for an experiment	30
Iteration	Experiments can be started at small allocation and ramped up to larger all. in iterations	29
Isolation	Concurrently run experiments can be configured to not share users/devices	29
Configuration management	Experiment variant is configured in the experimentation system; configs are passed to the clients along with variant assignments	26
Rollout	Supports rolling out the winning variant to 100% of users	25
AA testing	Pre- and post-experiment A/A	18
Power analysis	Helping determine experiment allocation and duration	17
Alerting	Automatically alerts on significant negative metric movements	15
Drill-down capability	Automated ways to diagnose a specific metric's movement	14
Cross-experiment analysis	Collects and preserves sufficient metadata about past experiments for cross-experiment analysis	14
Near Real Time Alerting	Alerts are generated in near real time	10
Ship advice	Automatically analyzes the results and makes ship/no-ship recommendation	8
Interaction detection	Interactions between experiments running concurrently on the same set of users are detected	7
Auto-shutdown	Harmful experiments are autom. terminated	6

Table 17. Experimentation platform features and count of Product Organizations that implemented and integrated them (out of 44).

Based on our findings, it is evident that autonomous shutdown of harmful experiments, interaction detection, and ship recommendation are lacking from the majority of the platforms.

12.4.2.3 DM3: Experimentation Pervasiveness.

This dimension is concerned with the extent of experiment penetration in the software development organization. Products that are starting to adopt experimentation typically experiment with a limited amount of new features. In contrast, in mature products, every change including the smallest bug fixes is released to customers under an experiment. In our survey we measure the progress along this dimensions by asking for the fraction of features evaluated through an experiment (with options: less than 10%, between 10% and 50%, between 50% and 90%, and above 90%). The results visible in Figure 2 suggest that **six product organizations evaluate over 90% of their features through an experiment, whereas nine organizations evaluate only 10% or less.** We compared these data with the number of features in the experimentation platforms of our survey participants.

% of features experimented with	#ExP Features (AVG)
< 10%	6
10% - 50%	8
50% - 90%	9
>90%	11

Table 18. % of features experimented vs. ExP feature count.

As visible in Table 18 above - on average - organizations that integrated more features into their experimentation platform release a larger percentage of product features through an experiment, and vice versa. These findings support our experience that making experimentation more efficient by introducing automation to decrease the load on data scientists can improve companies' experimentation capabilities. We illustrate this with the experience of Microsoft's Bing product organization on Figure 35, showing that its experimentation growth was fueled (among other reasons) by

introducing new platform features (e.g. Isolation and Interaction detection). This growth, in turn, creates requirements for more advanced features.

Figure 35. The Growth of Experimentation at Microsoft Bing [6].

12.4.2.4 DM4: Engineering Team Self-Sufficiency.

This dimension is concerned with the level of independent experiment management within the feature teams. In our view, and as companies evolve, experimentation should become a standard operating procedure for most engineering and product management professionals, which contrasts the early stages of experimentation where most of the OCEs are executed by data scientists or analysts with experimentation knowledge. Based on our data analysis, the majority of product organizations fall in the “Walk” stage (17 out of 44) along this dimension. In these organizations Engineers/Program Managers configure and start the experiment on their own, however, analysts help compute, analyze and interpret the results of the experiments. Furthermore, in 8 product organizations analysts only supervise the analysis. 8 out of 44 product organizations classify in the “Fly” stage, meaning that they conduct most experiments end-to-end without any analyst involvement. Similarly, as with the previous dimensions, there is a correlation between the amount of functionality implemented in the exp. platform and companies’ ability to self- experiment.

Despite the fact that evolving DM1 impacts DM4, there is another important factor that impacts the growth of DM4 – education. Experienced analysts at Microsoft frequently organize tutorials, conferences, workshops and co-analysis sessions to train individuals in the product teams to become better and more accurate in analyzing experiments. And although this may be considered time consuming and expensive from the resource perspective, there are many pitfalls described in [189] that suggest to take the abstraction of experimental methodology with care, especially when only limited level of training is given to product teams. Our suggestion for companies in the early stages of this dimension is to jointly advance DM1 and DM4 by providing trainings to product teams on how to run trustworthy experiments with the functionality that is incrementally added to the platform.

12.4.2.5 DM5: Experimentation Team Organization.

This dimension describes how companies organize their experimentation teams. Our data visible on Figure 2 reveals that the ‘center of excellence model’ where experimentation experts (e.g. engineers and analysts working on the experimentation platform) work together in enabling other product teams across the organization run experiments when needed is the most common organization of the experimentation team (21 out of 44). Respectively, 7 product organizations classify in the “Crawl” stage, 11 in the “Walk”, and 5 in the “Run” stage. In contrast to other dimensions, the progression along this dimension does not necessarily imply that a product organization is more or less mature for every type of company. In fact, having a centralized team responsible for experimentation without a clear strategy introduces new challenges to the equation, e.g. the lack of communication and expertise sharing illustrated by quote:

“Central teams are not communicating well with the individual teams and individual teams not having the required expertise.”

--PO33

In some organizations, for example in those with multiple businesses, it may be very challenging to aim for a centralized top-down strategy that dictates a center of experimentation excellence before the overall company is aligned and accepts to this path. In such scenario, multiple standalone teams for experimentation knowledge could be more efficient. In our research, we have not collected information about the type of the business that product organizations operate, neither in how many verticals they work. Our findings for this dimension are therefore merely informative and should be taken into consideration with the context of the company.

12.4.2.6 DM6: Overall Evaluation Criteria.

This dimension is concerned with the development and improvement of the key metrics that are used in experiments. In our experience and based on past case studies [16], [9] agreeing on a set of metrics which the overall product organization is optimizing for in experiments is critical. In prior research, several good and bad options for such metrics have been identified [122], [184], [220]. In our analysis of results visible on Figure 2 it is evident that only 2 out of 44 organizations succeeded in creating mature metrics that capture the essence of their business, and that can be used in experiments. Out of the collected responses, 21 product organizations conduct experiments with experiment specific metrics. Optimizing with ad-hoc metrics typically results in conflicts, or worse, in experiments that produce significant results by cannibalizing each other. In our view, this dimension of experimentation growth remains the most challenging, and more research is needed in order for product organizations to evolve. We support our reasoning with illustrative quotes captured through our online survey when asked for the greatest challenges:

“Experimentation results are not translating to Business impact. ”

--PO41

“Lack of aligned success criteria across the ecosystem.”

--PO44

“Lack of good OEC for most of experiments”

--PO27

“... sometimes key metrics are not well defined in a new type of features”

--PO26

12.4.2.7 DM6: *Experimentation Impact.*

The final dimension of the experimentation growth is concerned with the level of impact that experiments have on the product organization. Eventually, the impact of OCEs should extend beyond deciding between the few variants of a software feature within a team that works on a product. For example, as OCEs reveal the exact amount of contributions to the key metrics, feature teams’ success can be accurately measured and rewarded. Also, infrastructure capacity can be better planned and the quality of the product improved [25]. Only 5 product organizations report that they measure the success of the company through comprehensive OCE metrics, and that they reward their feature teams based on these improvements, or discuss the results revealing the understanding of their customer preferences with top executives. 17 product organizations reported that they run experiments in multiple teams, whereas only 3 organizations run experiments and experiment their benefits within a single feature team. Based on our experience, we believe that the reason why many companies do not experience the full benefits of experimentation is due to a relatively immature DM5. Without a well-defined OEC it is very difficult to measure the success of a product organization and provide rewards for it.

12.4.3 Key Findings and Challenges

Findings presented in the previous section reveal insights into how companies organize their experimentation teams, how mature their experimentation platforms are, and how many product changes result in a controlled rollout through an OCE. In this section we briefly reflect on our findings and list the key findings and challenges that we see with the current state.

12.4.3.1 *Key Findings:*

In-house experimentation platforms are the norm. Engineering an in-house platform that integrates well with the product is the common pattern.

Platform features are critical for EG score. Companies that integrated more features in their platform achieved a greater level of experimentation when compared to those that developed fewer features in their platform.

Analyst Supervision in Experimentation is the norm. Most of the companies rely on data scientist skills for experiment management and analysis supervision.

Experiments have a cross-team impact. Most companies report the capability to run experiments in multiple teams, impacting product roadmap and quality of the product.

12.4.3.2 *Key Challenges and Research Opportunities*

We list the key challenges by reflecting on the findings from the previous section, and the thematic coding analysis of the open-ended questions in our survey.

Defining and Agreeing on Key OEC Product Metrics. Despite the efforts[14], [15], developing the key metrics that efficiently measure the value of a software product remains one of the key challenges. As OECs are one of the best kept secrets of companies (and closely relate to business KPIs), it is unlikely that companies will share them, however, further research work on how to design OCEs is possible.

Engineering advanced features in the platform. Although implementing features such as randomization or treatment assignment might be clear, many features in our list require a profound understanding of experimentation. Further research is needed on how to architecture an experimentation platform that can be support OCEs at scale.

Hiring and Educating Experimentation Scientists. Recruitment for development teams is one of the most frequent challenges expressed by our survey participants. In fact, skills for experimental design have been mentioned as one of the most critical demands for Data Scientists in a recent survey [18], [133]. Many universities around the world have already adjusted their programs in order to educate and produce more Data Scientists, and we should consider having experimentation skills as a part of their curriculums.

Fostering Exp. Culture and Sharing of Learnings. Institutionalizing learnings from OCEs in an “experience factory” fashion [216] remains is open challenge. When the number of experiments grows beyond a handful a year, there is less scrutiny and involvement of the organization, and OCE findings remains within the feature team that ran the experiment. More research is needed on how to efficiently share learnings in the context of experimentation.

12.5 Conclusion and Future Work

Several large scale online companies such as Google, Microsoft, and Amazon frequently report on the benefits that they experience due to the growth of the experimentation practices in their companies [25], [47], [56]–[58]. However, the capabilities of the remainder of the software industry with respect to experimentation remain unknown [4], [10], [36], [59]. In this Chapter, we conducted an online empirical survey and learned what the experimentation maturity of the software industry that already conducts experiments is, and which are the challenges that prevent those companies from evolving to the next stage. Most companies that responded to our survey classify within the “Walk” and “Run” stage, meaning that they are able to conduct tens or hundreds of OCEs yearly. By investigating further, we learned that there is large variation along the seven dimensions of experimentation, which we presented in greater detail in section IV. We hope that companies running experimentation can use the survey results as a baseline for comparison towards the remainder of the industry, and researchers to steer their future work within the area of online experimentation. We are leaving the online survey open and continue to solicit interests from companies for providing their input for future research. Also, we aim to improve our online service to enable historical measurement of experimentation maturity, to better measure how and when companies move to the next stage.

13DISCUSSION

Delivering value to customers is paramount for the success of any company [6]. Software companies have always strived to learn what their customers value through interviews, questionnaires, focus groups, etc. [52], [62]–[66], [90], [182], [191]. Traditionally, however, feedback collection techniques have focused on asking the customers what they *say they do* with the product, and what they *say they like* or *dislike* with it [70], [182]. Today, the software industry is evolving towards complementing the learning process about the customers’ preferences with the use of data that reveals what their customers *actually do* with the product [10], [59], [183]. Product usage data [21], [23], [33], [34] such as telemetry or product logs can enable software companies to be more accurate in evaluating their ideas, and to move away from assumptions and towards trustworthy data-driven decision making through experimentation[29], [34], [55]. Effectively learning from data and building experimentation capabilities, however, has proven to be very challenging[9], [10], [131]. The list of possibilities to learn from customers is long, the list of pitfalls and caveats while doing this even longer [16], [26].

In the remainder of this chapter, we discuss the main contributions of this thesis with respect to our research questions and reflect on our findings.

13.1 Research Questions and Contributions

In Chapter 1.3.1, we introduced our research questions. Here, we summarize how we answered them in the preceding chapters. We start with an exploratory investigation of the different types of data handling techniques (RQ_{A1}) and its use (RQ_{A2}). Next, we dive into on one of the techniques – experimentation. Specifically, we describe how to adopt (RQ_{B1}), evolve (RQ_{B2}), and make experimentation effective RQ_{B3} in large-scale software development. Finally, we discuss the maturity of experimentation in practice (RQ_{B4}).

13.1.1 RQ_{A1} : How do software companies collect customer feedback and product data?

Our first research question is explorative and concerned with the general process of collecting customer feedback and product data, and its use. In our research, presented in chapter 4, we identified the different roles that conduct the collection and the different techniques employed for this purpose. In our research, we differentiated between three stages of product development. Namely, the pre-development stage, the development stage, and the post-deployment stage. In each of the three stages, we recognize a number of roles (e.g. product managers, Software Engineers, Release Managers, etc.), a number of different techniques that practitioners employ in order to learn from customers (e.g. reading of industrial reports, analyzing incidents, experimentation) and the types of data that are collected (e.g. incident duration, product logs, etc.).

We summarize the identified techniques, types of data collected, and the roles of practitioners that are involved in this process in Table 19.

	Collectors	Techniques	Feedback Data Types
Pre-Development	Strategy specialists, Product managers, Product owners	Reading of industry press, Reading of standards, Reading of internal reports, Reading customer visit reviews	Customer wishes, Short/Long market trends, Competitors ability of delivering the product
	Strategy specialists, Feature owners	Telephone interviews, Face-to-face interviews, Conducting group interviews	Existing product satisfaction, Future product specification, Personas and User Journeys
Development	UX specialists, Software Engineers	System Usability Scale Form, Asking open ended questions, Demonstrating prototypes, Filming of users' product use	Acceptance of the prototype, Eye behavior and focus time, Points of pain, Bottlenecks and constrains, Interaction design sketches
	System managers, System architects, Software engineers	Consolidate feedback from other projects, Reading prototype log entries	Small improvement wishes, Configuration data, Product operational data
Post-Deployment	Release managers, Service managers, Software engineers	Reading of customer reports, Analyzing incidents, Aggregating customer requests, Analyzing product log files, Experimenting	Number of incidents., Duration of incidents, Product operational data, Product performance data
	Sales managers	Reading articles in the media, Sentimental analysis, Customer events participation, Reading industry press, Performing trend analysis	Opinions about the appeal of the product, Performance of the product, Business case descriptions

Table 19. Feedback Collectors, Techniques, and Feedback Data Types.

We recognized over two dozens of techniques used by practitioners to collect various types of feedback data. In addition to the techniques themselves, we recognize a common pattern identified in our prior literature review [59], and in other prior publications (see e.g. [30], [40], [46]). The role of quantitative feedback as the product development stages evolve increases. In the early stages companies rely heavily on the qualitative feedback obtained through reading reports, asking customers through interviews, etc. When the product is in the hands of customers, operational data collected through product instrumentation or other channels is increasingly gaining momentum in informing the practitioners on the value of their product.

Along to recognizing the collectors and techniques, we identified a series of challenges that companies face while collecting and benefiting from data. One of those challenges is that the use of feedback lacks structure. Feedback data that is collected, does not succeed in being shared across the company in a way that others could benefit from it. We illustrate the challenges in the flow of feedback in a conceptual model in chapter 4. This disconnect creates a cycle where product features are becoming optimized based on what can easily be measured, instead of on what the customers value at the time. As we demonstrate later in chapter 8, sharing only a limited set of learnings can have profound implications in how metrics for measuring the value of a product is updated, and how the development of the product continues. More research in integrating approaches such as Experience Factory [216] or learnings from Experience Management [240] with novel data collection and analysis practices is needed in order to enable software companies to benefit from the full potential of the data that they collect.

13.1.2 **RQ_{A2}**: How can software companies use the feedback data that they collect to effectively realize software features?

In our second research question, we address the use of feedback data in greater detail. We recognized two high-level uses of feedback data. First, companies can use feedback to *differentiate* among the software features that they develop and prioritize the development activities based on this. Second, feedback data can be used to *efficiently measure* and impact the value of software features as they are used by the customers. Both uses of feedback data are presented in greater detail in chapter 5 and chapter 6.

The notion of differentiating among software features is not novel (see for example the search on the Kano model [78] or the 3LPM model [21]). Instead, and what we contribute with our work, is a complementary model to identify the different types of features developed in large-scale systems, and recommend actions on how to prioritize the development activities for each of the types. In summary (see chapter 5 for more details), we identified

four distinct types of features developed by large-scale software companies. First, we recognize “Duty” features. These are the components of a product that are required by policy makers and regulators. Second, we recognize “Wow” features. These are the features that the product is being sold for, however, are rarely used by customers. Next, we recognize “Checkbox” features - the features that are needed in products because the competitor has them in their portfolio, however are not the decisive factor to select one product over the other. Finally, we recognize the “Flow” features, which are the components of the product most exposed to the users of the product.

In our model presented in chapter 5 and visualized on Figure 36, we recognize characteristic points that enable the practitioners to differentiate between the four feature types. The characteristics are; *the stakeholder* (e.g. the requestor of the feature), *feature engagement* (e.g. the expected level of feature usage), *the source of the feedback* (e.g. the stakeholder generating the most feedback), *the focus of the feature* (e.g. is the feature indenting to minimally satisfy a known need, or to innovate in a new area?), its *impact on driving sales* (is it a feature focusing on the customer paying for the product?) and the *deployment frequency*. Based on the outcome of the differentiation process, companies can prioritize their development activities accordingly.

Figure 36. Feature Types Model.

In fact, by evaluating our model, we learned that ‘Duty’ and ‘Checkbox’ features development follows a common process. We named this approach Output-Driven Development in order to stress that for these two types of features, companies spend a considerable amount of time studying the requirements and/or competitors (typically in the pre-development stages of product development). In contrast, “Wow” and “Flow” features are being developed in a value-driven approach which we label as “Outcome-Driven Development”. Here, feature ideas are expressed as hypothesis about their value, and considerable amount of time is spent in evaluating whether and to what extent the hypothesis holds. Typically, and as we show in chapter 6, the hypotheses are based on metrics and evaluated throughout the lifecycle of a feature.

One of the most accurate and effective approaches for hypothesis testing is controlled experimentation. In the remainder of this chapter, we discuss this approach in greater detail by elaborating on how companies adopt it in order to evaluate their “Wow” and “Flow” features.

13.1.3 **RQ_{B1}** : How can software companies adopt and evolve experimentation?

One of the key questions frequently asked in software organizations is how do changes to products impact the product and the customers using it. To infer causality and conclude that the changes are the reason for the change, companies need to conduct experiments. In fact, many of them. In chapter 7, we introduce the notion of experiment lifecycle, and discuss in detail how to enable experimentation in large-scale software organizations in chapter 8. One of our main contributions to this research question is the Experimentation Growth Model (EGM), which we inductively derived from analyzing the experience of growing experimentation activities in over a dozen Microsoft products and further detailed through case studies at Skyscanner, Booking, and Intuit.

The EGM model (see Figure 37) depicts experimentation growth as four stages of evolution, starting from *Crawl* where experimen-

tation is ad-hoc, time-consuming, and provides limited value, to *Walk* to *Run* to *Fly*, where experimentation is integral part of every aspect of product development, enabling data-driven decisions at scale. ‘Crawl’ is the initial stage where software companies start to experiment, which typically happens within a single feature team and without an experimentation platform. In the ‘Walk’ stage, companies develop the capability to conduct tens of experiments per annum. In the ‘Run’ stage, multiple feature teams conduct hundreds of experiments yearly with an experimentation platform that controls for common pitfalls and alerts experimenters for bad experiments. Finally, in the ‘Fly’ stage, software companies are able to conduct thousands of OCEs yearly, evaluating not only new features, but also every bug fix through experiments for unexpected outcomes.

In the EGM model, the evolution of experimentation from one stage to the next advances along the seven most critical dimensions DM1-DM7:

- DM1: **‘Technical focus of product development activities’**: focuses on the engineering work (for example, building a reliable data pipeline).
- DM2: **‘Experimentation platform capability’**: focuses on the features that companies need in their experimentation platform.
- DM3: **‘Experimentation pervasiveness’** assesses the extent to which experimentation is being conducted in software companies (for example, from a few user interface experiments towards experimenting with every change).
- DM4: **‘Feature team self-sufficiency’**: assesses the extent to which individual feature team members manage most of the experiments without the involvement.
- DM5: **‘Experimentation team organization’**: describes how to organize the experimentation experts in each of the four stages, in order to experience the most benefit from conducting OCEs (for example, from no experimentation team to a center of excellence).
- DM6: **The ‘Overall Evaluation Criteria (OEC)’**: entails how the key metrics are defined (for example, from a few signals to a structured and well-defined set of metrics that are indicative of long term success of a change which is measured in a short-term experiment).
- DM7: **‘Experimentation Impact’**: reveals the extent to which the experimentation impacts the software company at each of the four stages (for example, from an impact to a small feature, towards an impact that sets strategic and team goals).

The seven dimensions of Experimentation Growth Model are in detail described in chapter 8, and briefly visualized on Figure 37.

	Crawl	Walk	Run	Fly
Technical focus	Data Collection: From <i>debug logging to contextual logging</i> .	Standardization. Creating a standard set of metrics: Success metrics, debug metrics, guardrail metrics, data quality metrics.	Comprehensiveness. Comprehensive data collection, reliable data pipeline, and an exhaustive set of metrics. Automated analysis of experiments.	Consolidation. Unified data pipeline enables easy creation of cross-team metrics. Standardized process for metric creation, evaluation and management.
Experimentation platform capability	None: No Experimentation platform needed.	Basic. 3rd party platform can be used or internally developed. Basic features such as Trustworthy Treatment Assignment, Power Analysis, Pre-Experiment A/A testing, are required.	Evolved. The experimentation platform should be extended with new features supporting larger volume of experiments, such as Alerting, Control of carry-over effects, Experiment iteration support, and metric creation. Teams manually record the most important learnings from experiments.	Advanced. Intelligent platform supporting features such as interaction control and detection, near real-time detection and automatic shutdown of harmful experiments, drill-down capabilities to understand metric movement and metric operations. Learnings from experiment are automatically recorded.
Experimentation pervasiveness	Visibility: A handful of experiments to showcase the value of experimentation	Scope. Broadening the types of experiments on a selected set of features (e.g. from design to performance, from performance to infrastructure).	Expanding. Experiments are conducted on the majority of new features developed for the product.	Complete. Experiment with all changes, even bug fixes.
Feature team self-sufficiency	Limited: Feature teams rely on Data Scientist to setup, manage and analyse experiments.	Experiment setup: Creating the experiment (instrumentation, A/A testing, assigning traffic) is managed by the feature team.	Experiment management: Feature team is responsible for monitoring experiments, making ramp-up and shut-down decisions, designing and deploying experiment-specific metrics.	Analysis. No Data Scientist involvement needed for most experiments. Experiment results are intuitive for feature team to interpret and make decisions.
Experimentation team organization	None. No experimentation team.	Standalone. Centralized experimentation team is created. No or very little experimentation skills in feature teams.	Embedded. Experimentation team data scientists are embedded into individual feature teams to become domain experts and grow experimentation in the team.	Partnership. Feature teams have their own experimentation experts that create a strong unity with business. Experimentation team serves as a center of excellence partnering with experimentation experts on feature teams.
Overall Evaluation Criteria (OEC)	Experiment Specific. OEC is defined for every experiment.	Defined. OEC is defined as a structured set of metrics consisting of Success, Guardrail and Data Quality metrics.	Evolved. OEC is validate over a large number of experiments and evolved with the findings from the learning experiments. Single metric as a weighted combination of others is desired.	Stable. Only periodic changes are allowed to an OEC (e.g. 1 per year). It is used to set the performance goals for teams within the organization.
Experimentation Impact	Feature Constrained to the team conducting incremental feature update.	Team. Starting to impact the team's activity planning (e.g. release cycle). The impact is shared within the larger team.	Multiple teams. Experimentation impacts infrastructure planning, product quality and product complexity (e.g. controlled feature removal).	Executive and External. The impact of experimentation is widespread across the product. Product teams are rewarded for experiment metric gains. Experiment results are shared with top executives, Board of Directors, and externally.

Figure 37. Experimentation Growth Model.

13.1.4 RQ_{B2} : How can software companies improve their experimentation capabilities?

EGM model summarized in the previous RQ depicts the path that was taken by our case companies to evolve their experimentation capabilities. It has been adopted by companies worldwide to evaluate the maturity of their experimentation capabilities. Progressing along each of the seven dimensions, however, is far from intuitive. Among others, it requires from companies to invest in standardizing data collection practices, develop and integrate an experimentation platform, organize data science teams, implement efficient experiment analysis tools etc. We addressed a subset of these challenges in chapters 9 and 10, where we studied DM2 and DM3 of the EGM model. Specifically, our goal was to create an understanding of how to engineer an experimentation platform that can be used at large-scale, and furthermore, how to effectively analyze the results of the experiments. Our main contributions that address RQ_{B2} are the architecture of an experimentation system used at Microsoft and presented in chapter 9, and a development, integration and evaluation of a tool that improves the analysis of experiments presented in chapter 10.

13.1.4.1 Experimentation Platform

The four core components of an experimentation platform are *experimentation portal*, *experiment execution service*, *log processing service*, and *analysis service*.

Experimentation portal is an interface between the experiment owner and the experimentation system, enabling the owner to configure, start, monitor and control the experiment throughout its lifecycle.

Experiment execution service is concerned with how different experiment types are executed. For example, experimentation execution service assures that experiment variants are delivered in the way that the system was configured. This can be done in three different ways which we described in greater detail in chapter 9.

Log processing service is responsible for collecting and cooking the log data and executing various analysis jobs. For example, delayed logs that arrive to the system are merged by this service with previously arrived logs.

Analysis service is used throughout the whole experiment lifecycle, informing experiment design, determining parameters of experiment execution, and helping experiment owners interpret the results. We examined this service in greater detail in chapter 10.

We visualize the experimentation platform with the aforementioned components on Figure 38.

Figure38. Architecture of an Experimentation Platform.

13.1.4.2 Effective Experiment Analysis

In addition to the experimentation platform, and to provide in-depth insights into one of the components of this platform, we studied the Experiment Analysis Services in greater detail. In order to effectively (accurately and timely) analyze experiments, it is critical to have tools in place that enable practitioners to learn about the **trustworthiness** of the experiment (were there any data quality issues that skew the results?) and its **outcome** (has the proposed change improved the metric aimed to improve?).

As we demonstrated in chapter 8, in order to evolve along the DM3: Experimentation Pervasiveness dimension of the EGM model, companies should invest into automation and tools for effective analysis of experiment data. One of the common approaches in experimentation to present analysis results is the balanced scorecard approach [177] visible on Figure 39 below (in this case indicating that the Overall Click Rate metric did not move due to statistically significant change in the Web Results metric).

Metric	T	C	Delta (%)	p-value
Overall Click Rate	0.9206	0.9219	-0.14%	8e-11
Web Results	0.5743	0.5800	-0.98%	~0
Answers	0.1913	0.1901	+0.63%	5e-24
Image	0.0262	0.0261	+0.38%	0.1112
Video	0.0280	0.0278	+0.72%	0.0004
News	0.0190	0.0190	+0.10%	0.8244
Related Search	0.0211	0.0207	+1.93%	7e-26
Pagination	0.0226	0.0227	-0.44%	0.0114
Other	0.0518	0.0515	+0.58%	0.0048

Figure 39. Experiment Scorecard.

However, and as we demonstrate in chapter 10, by complementing the balanced scorecard with our tool, companies that conduct experiments can make the experiment analysis process more effective. In our work presented in chapter 10, we studied in greater detail the experiment analysis process, and we developed, integrated and evaluated a tool that improves this process. Our tool (a mockup of it is visible on Figure 40 below) displays the most critical experiment data in a two dimensional visualization following the common experiment-analysis pattern that starts with data-quality inspection and continues with key success metric examinations (OEC). Our tool improves the *velocity* of experiment analysis (the effort spent until a certain insight is discovered) and its *comprehensiveness* (the number of insights and their correctness).

By using our tool experimenters are guided from a ‘birds-view’

Segment	Data Quality View	OEC View	SOI View
*Aggregate	Aggregate	Aggregate	10
Gender	M F ?	M F ?	2 2 2
Device Type	M W ?	M W ?	1 2 1
Data Center	1 2 3 4 5	1 2 3 4 5	1 2 1 2 8
Location	US	US	10 1

Figure 40. Effective Experiment Analysis.

perspective to the experiment segment slices (e.g. a certain age group), where the likelihood of finding relevant experiment findings is the highest. Based on an evaluation that we conducted through a controlled experiment with over 20 practitioners through 86 experiment analysis instances (see chapter 10.5 for details), we confirmed that our tool improves the experimentation capabilities of software companies by increasing the accuracy and velocity of experiment analysis.

13.1.5 *RQ_{B3}*: How evolved are the experimentation capabilities of software companies, and what are the benefits of becoming data-driven at scale?

We answered this research question in chapters 11 and 12, where we outlined the benefits of experimentation, and the current state of experimentation evolution. We learned in chapter 11 that by conducting experiments, companies (1) very accurately discover and (2) validate what their customers value, that (3) experiments increase the quality of the product that is being released, that (4) significant savings can be achieved on infrastructure resources when running experiments, and that work efforts of teams can be better (5) planned and (6) measured. The benefits of experiments therefore stretch well beyond just the common help in deciding between product variations identified in prior research [37], [154]. With experimentation, companies build a learning organization that is very difficult to sidestep and very quickly becomes business critical.

Far from all of the companies, however, experience all of the benefits that we identified in our research. Based on findings in chapter 12 where we surveyed 44 product organization for their experimentation maturity status, we can conclude that majority of the surveyed companies that already conduct online controlled experiments fit into either “Walk” or “Run” stages of our EGM model. Moreover, when we analyzed the detailed breakdown (see Figure 41 and Chapter 12 for greater detail), we learned which are the dimensions that require the most emphasis in our future research. We briefly highlight these on Figure 41 below, illustrating

how the surveyed organizations classify with respect to each of the dimensions of our Growth Model.

Figure 41. Experimentation Maturity Breakdown for every dimension of the Experimentation Growth Model (n=44).

DM2 (Experimentation Platform maturity) and DM6 (OEC maturity) are the least evolved dimensions among the surveyed companies. For example, only two product organizations out of 22 have fully evolved the experimentation platform. Our contribution in chapter 9 is directly addressing DM2 and aiming to provide guidelines of how to engineer an experimentation platform. However, future work is needed on improving DM6 in order to help companies design more informative metrics. Specifically, metrics that can be used as an Overall Evaluation Criteria that correctly captures the value of a software products. Although many metrics can be expressed in a way that we presented in chapter 6, OEC metrics should be broad and sensitive to changes.

13.2 From Data to Data-Driven Companies

In the previous sections, we discussed the answers to our research questions. In this section, we reflect on our work and contributions by describing on a high-level what we consider as data-driven companies.

Based on our accumulated knowledge, and after spending several years exploring and improving data-driven capabilities of large-scale software companies, we define the concept of Data-Driven Companies (DDC). In our view, Data-Driven Companies can be considered as **experimental laboratories** that employ **scientific trustworthy procedures** to accurately, *iteratively reproducible*, and *completely* utilize data in order to **continuously improve** their software products and data handling procedures. We elaborate briefly on each of the highlights next.

13.2.1 Accuracy

With *accuracy*, we refer to the selection of the operational procedure for collecting and analyzing the data, and the confidence that gets assigned to the findings when using the procedure. As we showed in chapter 4, there are a number of different techniques that companies employ today to collect feedback. The selection of the feedback collection and analysis method depends on the purpose of the task, the capabilities of the company, and can vary throughout the product development lifecycle. DDC are aware of the limitations and biases of the data-handling methods that they employ.

In addition, and as demonstrated in chapters 8-10, DDC continuously scrutinize their data operating procedures and try to identify flaws in them. For example, when experimentation is being used, pre-experiment A/A tests [9], [14], [121] are being conducted before the experiment starts to select a distribution that is most balanced, post-experiment AA analyses are done after the experiment is completed [9], [14], [121], and data-quality issues are investigated prior to the examination of experiment results. When novelty [57], [192] or primacy effects are detected, these are recognized and highlighted. In summary, all aspects that impact the ac-

curacy of the findings on the data are considered while conducting decisions. This is achieved through education and automation.

13.2.2 Iterative and Reproducible

DDC conduct multiple rounds of learning throughout the different stages of a product lifecycle, leading to reproducible results. For example, and as shown in chapters 4-7, DDC use data to make decisions in the pre-development stage of the product, as well as in the development and post-deployment stages. In the pre-development stage, the data is mainly used to build hypotheses about the value of the features being developed, and to differentiate among the different types of software features. We demonstrate this in chapters 5 and 6. In the post-deployment stage, and as demonstrated in chapters 6-12 the data is used to test those hypotheses, or to learn about unexpected consequences that form new ideas.

When the evaluations are repeated, the results need to be reproducible in order for them to be trustworthy. For example, if a controlled experiment was executed on a set of customers, any next experiment on another representative set needs to result in nearly identical outcome. At the same time, DDC are aware that customer preferences change over time and that proven hypotheses can expire. Therefore, metrics that are being used to measure the success of their product changes are under constant scrutiny and exposed to a similar continuous improvement process as the product features itself.

13.2.3 Completeness

With *completeness*, we aim to stress that DDC conduct decisions only when the complete data, that can impact the outcome, has been analyzed. In chapter 10, we illustrate in detail on how to effectively analyze data through experiment analysis. However, the analysis part is only one component in the data chain. To increase the completeness, an embedded awareness of data-driven development is required in many different roles across the organization. Roles that both design and perform the data collection activities, as well with the analysts in charge of the data analysis process. We illustrate in chapter 8 the importance of organizing practitioners at various levels of maturity, and how to evolve the data collection and analysis practices.

Similarly as scientific publications require triangulation of data collection, so do the product development data activities. As we learned in our research and demonstrated in chapter 4, even the most impactful methods such as experimentation benefit qualitative data to become efficient. Also, the completeness relates to accuracy; any analysis of data in DDC requires a thorough examination of both raw feedback that was collected (e.g. for outlier detection), as well as the computed results. For example, in experimentation, power analysis needs to be conducted in order to determine the number of experiment participants in needed for a certain level of statistical significance, and significantly larger.

13.2.4 Continuous Improvement

With *improvement*, we stress the need for continuous evolution of data-driven practices. This aspect has two dimensions.

First, DDC automate the steps that need to be done when using a data collection and analysis procedure. In chapter 9, for example, we present how to automate the most critical components of the experimentation process by engineering an experimentation architecture. Automation impacts reproducibility as it enables not only faster, but also more accurate interpretability of results. As we demonstrate in chapter 12, companies that conduct many experiments and benefit the most from the data that they collect, develop in-house experimentation platforms, and integrate them well with their products and processes.

Second, DDC companies adopt new methods and analysis possibilities to their toolbox as they mature. For example, DCC that collect operational data from their products, and are able to deploy configurations to them, can potentially introduce experimentation in their development process. At the same time, existing procedures are being improved. Through the improvement of data-driven capabilities, DDC experience benefits that span beyond determining a better version of a product. As we demonstrate in chapter 11, evolved DDC experience the benefits on the product, team and portfolio levels, enabling every individual in an organization to conduct more accurate decisions.

14 CONCLUSION

In this thesis, we explored how data-driven companies collect and use customer feedback and product data to better prioritize product development activities for the long-term value of the products that they develop. We investigated in detail how companies experiment with their products while they are in use, and we constructed an understanding of how the experimentation capabilities can be improved to become more effective at large scale.

In particular, we first explored the different ways that companies collect feedback and data from and about the products that they develop. Second, we examined how these data can be used to differentiate between the different types of features that are being developed, and how to prioritize the development activities for each of the different types. Next, we analyzed and described in detail how companies adopt one of the techniques for becoming data-driven - experimentation. We illustrated the most critical dimensions of evolving experimentation capabilities and addressed two of those dimensions (Experimentation Platform and Experimentation Pervasiveness) in greater detail. Next, we presented an architecture of an experimentation system used at one of our case companies and improved one of its components by developing an artefact and evaluated its effectiveness.

Finally, we described the benefits of becoming data-driven through experimentation at large scale and outlined the current state of experimentation maturity based on surveying over 40 companies that already run large-scale experimentation in practice.

With our work, we hope to empower and guide software companies in evolving their data-driven capabilities towards experimentation at large scale. Ultimately, this should lead to better products and services for all of us.

14.1 Future work

In our research, we identified and addressed several challenges that prevented companies from scaling their data-driven capabilities. Some of these challenges, however, require future work.

The first challenge that is likely to gain momentum in the upcoming years is “per-tenant experimentation”. Companies that develop products primarily bought by other companies (e.g. B2B org.) are sometimes not able to experiment due to a number of constraints. For example, an individual customer might require identical experience for all of their users. Or, the count of users to randomize on is too low (see for example the challenges in [241]). Such conditions prevent practitioners from randomizing within the same organization, making the traditional experiment analysis inaccurate. To enable such companies to experiment and scale, more knowledge is needed on how to guarantee statistical significance while preserving customer-specific experience.

Next, we see a number of research possibilities with respect to the EGM model and describing its dimensions in greater detail. In our research, we addressed in greater detail only two out of seven dimensions of Our EGM model visible on Figure 37. Our experience, however, is that maximum ROI from experimentation is derived when all dimensions are advanced simultaneously. Companies adopting or evolving their experimentation capabilities could therefore benefit from future research on how to advance along the other five dimensions of the EGM model. Specifically, and as identified in our survey on the current state of experimentation maturity in chapter 12.4.3.2, research in designing metrics that can be

used as Overall Evaluation Criteria requires critical attention from the companies that already experiment.

Finally, future work is needed on identifying accurate leading metrics to evaluate the progresses along the dimensions of the EGM model. Currently, the assessment can be done through qualitative questioning through our online tool [214], however, by complementing this with quantitative metrics (e.g. number of experiments ran without data scientist involvement for DM4), we could automate this process and make it more accurate.

BIBLIOGRAPHY

- [1] P. Diamantis, “Culture & Experimentation — with Uber’s Chief Product Officer,” *Medium*, 2016. [Online]. Available: <https://medium.com/abundance-insights/culture-experimentation-with-uber-s-chief-product-officer-520dc22cfc4>.
- [2] M. Khurum, T. Gorschek, and M. Wilson, “The software value map - an exhaustive collection of value aspects for the development of software intensive products,” *Journal of Software: Evolution and Process*, vol. 25, no. 7, pp. 711–741, Jul. 2013.
- [3] S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, and P. Grünbacher, *Value-Based Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006.
- [4] F. Fagerholm, A. S. Guinea, H. Mäenpää, and J. Münch, “The RIGHT model for Continuous Experimentation,” *Journal of Systems and Software*, vol. 0, pp. 1–14, 2015.
- [5] M. Denne and J. Cleland-Huang, “The incremental funding method: Data-driven software development,” *IEEE Software*, vol. 21, no. 3, pp. 39–47, 2004.
- [6] B. Boehm, “Value-based software engineering: reinventing,” *SIGSOFT Softw. Eng. Notes*, vol. 28, no. 2, p. 3–, 2003.
- [7] M. Fowler, “Continuous Integration,” *Integration The Vlsi Journal*, vol. 26, no. 1, pp. 1–6, 2006.
- [8] D. G. Feitelson, E. Frachtenberg, and K. L. Beck, “Development and deployment at facebook,” *IEEE Internet Computing*, vol. 17, no. 4, pp. 8–17, 2013.
- [9] A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch, “The Evolution of Continuous Experimentation in Software Product

- Development: From Data to a Data-Driven Organization at Scale,” in *Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, 2017, pp. 770–780.
- [10] E. Lindgren and J. Münch, “Raising the odds of success: The current state of experimentation in product development,” *Information and Software Technology*, vol. 77, pp. 80–91, 2015.
 - [11] R. Kohavi and S. Thomke, “The Surprising Power of Online Experiments,” *Harvard Business Review*, no. October, 2017.
 - [12] R. C. Martin, *Agile Software Development, Principles, Patterns, and Practices*. 2002.
 - [13] K. Kevic, B. Murphy, L. Williams, and J. Beckmann, “Characterizing experimentation in continuous deployment: A case study on bing,” in *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track, ICSE-SEIP 2017*, 2017.
 - [14] R. Kohavi, A. Deng, B. Frasca, R. Longbotham, T. Walker, and Y. Xu, “Trustworthy online controlled experiments,” in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '12*, 2012, p. 786.
 - [15] P. Dmitriev, B. Frasca, S. Gupta, R. Kohavi, and G. Vaz, “Pitfalls of long-term online controlled experiments,” in *2016 IEEE International Conference on Big Data (Big Data)*, 2016, pp. 1367–1376.
 - [16] P. Dmitriev, S. Gupta, K. Dong Woo, and G. Vaz, “A Dirty Dozen: Twelve Common Metric Interpretation Pitfalls in Online Controlled Experiments,” in *Proceedings of the 23rd ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '17*, 2017.
 - [17] A. Deng, J. Lu, and J. Litz, “Trustworthy Analysis of Online A/B Tests,” *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining - WSDM '17*, pp. 641–649, 2017.
 - [18] M. Kim, T. Zimmermann, R. DeLine, and A. Begel, “The emerging role of data scientists on software development teams,” in *Proceedings of the 38th International Conference on Software Engineering - ICSE '16*, 2016, no. MSR-TR-2015-30, pp. 96–107.
 - [19] J. Bosch, “Speed, data, and ecosystems: the future of software engineering,” *IEEE Software*, vol. 33, no. 1, pp. 82–88, 2016.

- [20] H. Chen and V. C. Storey, "Business Intelligence and Analytics: From Big Data to Big Impact," vol. 36, no. 4, pp. 1165–1188, 2012.
- [21] H. Li, W. Shang, and A. E. Hassan, "Which log level should developers choose for a new logging statement?," *Empirical Software Engineering*, vol. 22, no. 4, pp. 1684–1716, 2017.
- [22] Q. Fu *et al.*, "Where do developers log? an empirical study on logging practices in industry," in *Companion Proceedings of the 36th International Conference on Software Engineering - ICSE Companion 2014*, 2014, pp. 24–33.
- [23] T. Barik, R. Deline, S. Drucker, and D. Fisher, "The Bones of the System: A Case Study of Logging and Telemetry at Microsoft," 2016.
- [24] A. Deng, P. Dmitriev, R. Kohavi, and P. Raff, "A / B Testing at Scale: Accelerating Software Innovation," *SIGIR*, pp. 1395–1397, 2017.
- [25] A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch, "The Benefits of Controlled Experimentation at Scale," in *Proceedings of the 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2017, pp. 18–26.
- [26] A. Deng, J. Lu, and J. Litz, "Trustworthy Analysis of Online A/B Tests: Pitfalls, Challenges and Solutions," in *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, 2017, pp. 641–649.
- [27] G. Schermann, J. Cito, P. Leitner, U. Zdun, and H. C. Gall, "We're Doing It Live: A Multi-Method Empirical Study on Continuous Experimentation," *Information and Software Technology*, Mar. 2018.
- [28] O. Rissanen and J. Munch, "Continuous Experimentation in the B2B Domain: A Case Study," *Proceedings - 2nd International Workshop on Rapid Continuous Software Engineering, RCoSE 2015*, pp. 12–18, 2015.
- [29] R. Kohavi and R. Longbotham, "Online Controlled Experiments and A/B Tests," in *Encyclopedia of Machine Learning and Data Mining*, no. Ries 2011, 2015, pp. 1–11.
- [30] H. H. Olsson and J. Bosch, *The HYPEX model: From opinions to data-driven software development*. 2014.
- [31] E. Ries, *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. 2011.

- [32] The Standish Group, "The standish group report," *Chaos*, vol. 49, pp. 1–8, 1995.
- [33] S. Marciuska, C. Gencel, and P. Abrahamsson, "Exploring How Feature Usage Relates to Customer Perceived Value: A Case Study in a Startup Company," in *Lecture Notes in Business Information Processing*, 2013, vol. 150 LNBIP, no. JUNE, pp. 166–177.
- [34] S. Marciuska, C. Gencel, and P. Abrahamsson, "Feature usage as a value indicator for decision making," *Proceedings of the Australian Software Engineering Conference, ASWEC*, pp. 124–131, 2014.
- [35] E. Backlund, M. Bolle, M. Tichy, H. H. Olsson, and J. Bosch, "Automated user interaction analysis for workflow-based web portals," in *Lecture Notes in Business Information Processing*, 2014, vol. 182 LNBIP, pp. 148–162.
- [36] A. Fabijan, H. H. Olsson, and J. Bosch, "The Lack of Sharing of Customer Data in Large Software Organizations: Challenges and Implications," in *Proceedings of the 17th International Conference on Agile Software Development XP2016*, 2016, pp. 39–52.
- [37] Y. Xu, N. Chen, A. Fernandez, O. Sinno, and A. Bhasin, "From Infrastructure to Culture: A/B Testing Challenges in Large Scale Social Networks," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015, no. Figure 1, pp. 2227–2236.
- [38] K. Pohl, *Requirements Engineering: Fundamentals, Principles, and Techniques*. 2010.
- [39] M. Chen, Y. Chang, and M. Chen, "How customer involvement enhances innovation performance: The moderating effect of appropriability," *Service Systems and Service ...*, pp. 1–6, 2014.
- [40] P. Bosch-Sijtsema and J. Bosch, "User Involvement throughout the Innovation Process in High-Tech Industries," *Journal of Product Innovation Management*, vol. 32, no. 5, pp. 1–36, 2014.
- [41] D. Pagano and B. Bruegge, "User involvement in software evolution practice: A case study," in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 953–962.
- [42] C. Bizer, P. Boncz, M. L. Brodie, and O. Erling, "The meaningful use of big data: Four perspectives - Four challenges," *SIGMOD Record*, vol. 40, no. 4, pp. 56–60,

- 2011.
- [43] T. Harford, “Big data: A big mistake?,” *Significance*, vol. 11, no. 5, pp. 14–19, Dec. 2014.
 - [44] P. Dmitriev and X. Wu, “Measuring Metrics,” in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management - CIKM '16*, 2016, pp. 429–437.
 - [45] H. H. Olsson and J. Bosch, “From opinions to data-driven software R&D: A multi-case study on how to close the ‘open loop’ problem,” in *Proceedings - 40th Euromicro Conference Series on Software Engineering and Advanced Applications, SEAA 2014*, 2014, pp. 9–16.
 - [46] H. H. Olsson and J. Bosch, “Towards continuous customer validation: A conceptual model for combining qualitative customer feedback with quantitative customer observation,” in *Lecture Notes in Business Information Processing*, 2015, vol. 210, pp. 154–166.
 - [47] J. Manzi, *Uncontrolled: the surprising payoff of trial-and-error for business, politics, and society*. Basic Books, 2012.
 - [48] G. Castellion, “Do It Wrong Quickly: How the Web Changes the Old Marketing Rules by Mike Moran.,” *Journal of Product Innovation Management*, vol. 25, no. 6, pp. 633–635, 2008.
 - [49] F. Fagerholm, A. S. Guinea, H. Mäenpää, and J. Münch, “Building Blocks for Continuous Experimentation,” *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*, pp. 26–35, 2014.
 - [50] J. Bosch, *Continuous Software Engineering*, vol. 9783319112. 2014.
 - [51] T. Savor *et al.*, “Continuous Deployment at Facebook and OANDA,” 2016.
 - [52] H. H. Olsson, H. Alahyari, and J. 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 - 38th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2012*, 2012, pp. 392–399.
 - [53] P. Rodriguez *et al.*, “Continuous Deployment of Software Intensive Products and Services: A Systematic Mapping Study,” *Journal of Systems and Software*, 2015.
 - [54] L. Chen, “Towards Architecting for Continuous Delivery,” *Proceedings - 12th Working IEEE/IFIP Conference on Software Architecture, WICSA 2015*, pp. 131–134, 2015.

- [55] H. Davenport, Thomas, "How to Design Smart Business Experiments," *Harvard Business Review*, vol. 0, 2009.
- [56] M. L. T. Cossio *et al.*, *A/B Testing - The most powerful way to turn clicks into customers*, vol. XXXIII, no. 2, 2012.
- [57] R. Kohavi, A. Deng, B. Frasca, T. Walker, Y. Xu, and N. Pohlmann, "Online controlled experiments at large scale," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '13*, 2013, p. 1168.
- [58] D. Tang, A. Agarwal, D. O'Brien, and M. Meyer, "Overlapping experiment infrastructure," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '10*, 2010, p. 17.
- [59] A. Fabijan, H. H. Olsson, and J. Bosch, "Customer Feedback and Data Collection Techniques in Software R&D: A Literature Review," in *Proceedings of Software Business, ICSOB 2015*, 2015, vol. 210, pp. 139–153.
- [60] W. W. Royce, "Managing the development of large software systems," *Electronics*, vol. 26, no. August, pp. 1–9, 1970.
- [61] P. Rook, "Controlling software projects," *Software Engineering Journal*, vol. 1, no. 1, pp. 7–16, 1986.
- [62] M. W. Alford, "A Requirements Engineering Methodology for Real-Time Processing Requirements," *IEEE Transactions on Software Engineering*, vol. SE-3, no. 1, pp. 60–69, 1977.
- [63] E. Horowitz and B. W. Boehm, *Practical strategies for developing large software systems*. Addison-Wesley Reading, Ma, 1975.
- [64] J. Kabbedijk, S. Brinkkemper, S. Jansen, and B. Van Der Veldt, "Customer involvement in requirements management: Lessons from mass market software development," *Proceedings of the IEEE International Conference on Requirements Engineering*, no. Section 2, pp. 281–286, 2009.
- [65] W. Maalej, M. Nayebi, T. Johann, and G. Ruhe, "Toward data-driven requirements engineering," *IEEE Software*, vol. 33, no. 1, pp. 48–54, 2016.
- [66] S. Marczak and D. Damian, "How interaction between roles shapes the communication structure in requirements-driven collaboration," in *Proceedings of the 2011 IEEE 19th International Requirements Engineering Conference, RE 2011*, 2011, pp. 47–56.
- [67] C. Avgerou, "Information systems: what sort of science is it?," *Omega*, vol. 28, no. 5, pp. 567–579, 2000.

- [68] R. Ling and D. C. Yen, "Customer Relationship Management: An Analysis Framework and Implementation Strategies," *Journal of Computer Information Systems*, vol. 41, no. 3, pp. 82–97, 2001.
- [69] J. Hess, S. Offenbergh, and V. Pipek, "Community Driven Development as participation? - Involving User Communities in a Software Design Process," ... *Conference on Participatory Design 2008*, pp. 31–40, 2008.
- [70] J. Hess, D. Randall, V. Pipek, and V. Wulf, "Involving users in the wild—Participatory product development in and with online communities," *International Journal of Human-Computer Studies*, vol. 71, no. 5, pp. 570–589, 2013.
- [71] J. Stark, *Product Lifecycle Management*. Cham: Springer International Publishing, 2015.
- [72] S. G. Blank, "The Four Steps to the Epiphany: Successful Strategies for Products that Win. San Mateo," CA: *Cafepress.com*, vol. 15, no. 2, 2005.
- [73] E. von Hippel, "Lead Users: A Source of Novel Product Concepts," *Management Science*, vol. 32, no. 7, pp. 791–805, 1986.
- [74] E. Bjarnason, *Integrated Requirements Engineering – Understanding and Bridging Gaps within Software Development*. 2013.
- [75] L. Williams and A. Cockburn, "Introduction: Agile Software Development: Its About Feedback and Change," *Computer*, vol. 36, no. 6. p. 39, 2003.
- [76] D. Bahn and J. Naumann, "Conversation about software requirements with prototypes and scenarios," *Next Generation Information*, vol. 2382, pp. 147–157, 2002.
- [77] R. B. Everhart, D. M. Fetterman, J. P. Goetz, and M. D. Le Compte, "Ethnography and Qualitative Design in Educational Research.," *Contemporary Sociology*, vol. 15, no. 3, p. 450, 1986.
- [78] D. Leffingwell and D. Widrig, "Managing software requirements: a unified approach," *Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA*, vol. 10, no. 4, p. 491, 1999.
- [79] S. Henninger, "Tool support for experience-based methodologies," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 2640, pp. 44–59, 2003.

- [80] Thomas Tullis, William Albert, T. Tullis, and B. Albert, "Chapter 3 - Planning a Usability Study," in *Measuring the User Experience; Collecting, Analyzing, and Presenting Usability Metrics Interactive Technologies 2013*, T. Tullis and B. Albert, Eds. San Francisco: Morgan Kaufmann, 2013, pp. 41–62.
- [81] K. Holtzblatt, J. B. Wendell, and S. Wood, "Chapter 12 - Storyboarding," in *Rapid Contextual Design*, K. Holtzblatt, J. B. Wendell, and S. Wood, Eds. San Francisco: Morgan Kaufmann, 2005, pp. 229–243.
- [82] D. Kutetsky and V. Gritsyuk, "KSU feedback service as a tool for getting feedback in educational institutions. perspectives of use in government organizations and commercial companies," in *CEUR Workshop Proceedings*, 2012, vol. 848, pp. 134–146.
- [83] K. Blumer, N. AlGharibeh, and M. Eid, "TOP: Tactile Opinion Poll System for Silent Feedback," *2014 Ieee International Symposium on Haptic, Audio and Visual Environments and Games (Have)*, 2014.
- [84] M. Almaliki, C. Ncube, and R. Ali, "The Design of Adaptive Acquisition of Users Feedback: an Empirical Study," in *2014 Ieee Eighth International Conference on Research Challenges in Information Science (Rcis)*, 2014.
- [85] N. Mack, C. Woodsong, K. M. McQueen, G. Guest, and E. Namey, *Qualitative Research Methods: A data collector's field guide*. 2011.
- [86] N. Kano, N. Seraku, F. Takahashi, and S. Tsuji, "Attractive quality and must-be quality," *Journal of the Japanese Society for Quality Control*, vol. 14, no. 2, pp. 39–48, 1984.
- [87] C. Ofir and I. Simonson, "In Search of Negative Customer Feedback: The Effect of Expecting to Evaluate on Satisfaction Evaluations," *Journal of Marketing Research*, vol. 38, no. 2, pp. 170–182, 2001.
- [88] B. Fitzgerald and K.-J. Stol, "Continuous software engineering and beyond: trends and challenges," *Fitzgerald, B., & Stol, K.-J. (2014). Continuous software engineering and beyond: trends and challenges. Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering - RCoSE 2014*, 1–9. doi:10.1145/2593812.2593813*Proceedings of th*, pp. 1–9, 2014.
- [89] B. Boehm, "Requirements that handle IKIWISI, COTS, and rapid change," *Computer*, vol. 33, no. 7, pp. 99–102, 2000.

- [90] L. Cao and B. Ramesh, "Agile requirements engineering practices: An empirical study," *IEEE Software*, vol. 25, no. 1, pp. 60–67, 2008.
- [91] X. Wang, K. Conboy, and O. Cawley, "'Leagile' software development: An experience report analysis of the application of lean approaches in agile software development," *Journal of Systems and Software*, vol. 85, no. 6, pp. 1287–1299, 2012.
- [92] C. Larman, *Scaling lean & agile development: thinking and organizational tools for large-scale Scrum*. Pearson Education India, 2008.
- [93] J. Liker, "The Toyota Way," *Clinical Medicine*, vol. 11, pp. 308–10, 2003.
- [94] E. M. Goldratt and J. Cox, *The Goal: A Process of Ongoing Improvement*, vol. 2nd rev. e, no. 337 p. 2004.
- [95] S. Mujtaba, R. Feldt, and K. Petersen, "Waste and lead time reduction in a software product customization process with value stream maps," in *Proceedings of the Australian Software Engineering Conference, ASWEC*, 2010, pp. 139–148.
- [96] T. Sedano, P. Ralph, and T. Sedano, "Software Development Waste," in *Proceedings of the 39th International Conference on Software Engineering - ICSE '17*, 2017, no. May, pp. 130–140.
- [97] M. STOICA, M. MIRCEA, and B. GHILIC-MICU, "Software Development: Agile vs. Traditional," *Informatica Economica*, vol. 17, no. 4/2013, pp. 64–76, 2013.
- [98] A. Lancaster, "Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces," *Ieee Transactions On Professional Communication*, vol. 47, no. 4, pp. 335–336, Dec. 2003.
- [99] J. Zhang and J. Chung, "Mockup-driven fast-prototyping methodology for Web application development," *Software: Practice and Experience*, vol. 33, no. 13, pp. 1251–1272, Nov. 2003.
- [100] J. Arnowitz, M. Arent, and N. Berger, "Chapter 14 - Card Sorting Prototyping," in *Effective Prototyping for Software Makers*, J. Arnowitz, M. Arent, and N. Berger, Eds. San Francisco: Morgan Kaufmann, 2007, pp. 250–271.
- [101] R. F. Ciriello and A. Richter, "When Prototyping Meets Storytelling: Practices and Malpractices in Innovating Software Firms," in *Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track*,

- 2017, no. May, pp. 163–172.
- [102] E. L. Wagner and G. Piccoli, “Moving beyond user participation to achieve successful IS design,” *Communications of the ACM*, vol. 50, no. 12, pp. 51–55, Dec. 2007.
 - [103] H. R. Hartson and E. C. Smith, “Rapid prototyping in human-computer interface development,” *Interacting with Computers*, vol. 3, no. 1, pp. 51–91, Apr. 1991.
 - [104] E. Lindgren and J. Münch, “Software development as an experiment system: A qualitative survey on the state of the practice,” in *Lecture Notes in Business Information Processing*, 2015, vol. 212, pp. 117–128.
 - [105] J. Evnin and M. Pries, “Are you sure? Really? A contextual approach to agile user research,” in *Proceedings - Agile 2008 Conference*, 2008, pp. 537–542.
 - [106] K. M. Nowack, “360-degree feedback: The whole story,” *Training & Development*, vol. 47, no. 1, pp. 69–73, 1993.
 - [107] D. Ståhl and J. Bosch, “Continuous integration flows,” in *Continuous software engineering*, vol. 9783319112, 2014, pp. 107–115.
 - [108] M. Shahin, M. Zahedi, M. A. Babar, and L. Zhu, “Adopting Continuous Delivery and Deployment,” *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering - EASE’17*, no. i, pp. 384–393, 2017.
 - [109] L. Chen, “Microservices: Architecting for Continuous Delivery and DevOps,” in *IEEE International Conference on Software Architecture (ICSA)*, 2018, no. March.
 - [110] D. Cannon, D. Cannon, D. Wheeldon, S. Lacy, and A. Hanna, *ITIL service strategy*. Tso, 2011.
 - [111] B. Fitzgerald and K.-J. Stol, “Continuous software engineering: A roadmap and agenda,” *Journal of Systems and Software*, p. , 2015.
 - [112] J. Humble and J. Molesky, “Why enterprises must adopt devops to enable continuous delivery,” *Cutter IT Journal*, vol. 24, no. 8, p. 6, 2011.
 - [113] S. G. Yaman *et al.*, “Customer Involvement in Continuous Deployment: A Systematic Literature Review,” in *Requirements Engineering: Foundation for Software Quality: 22nd International Working Conference, REFSQ 2016, March 14-17, 2016.*, 2016, pp. 249–265.
 - [114] W. Bank, S. Survey, S. P. Committee, and G. S. Office, “Exploratory data nalysis,” *Reading, Mass.*, pp. 5–24, 1997.
 - [115] D. Sculley, “Practical learning from one-sided feedback,” in

- Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '07*, 2007, p. 609.
- [116] J. W. Tukey, *Exploratory Data Analysis*, vol. 2, no. 1999. 1977.
 - [117] L. S. Aiken and S. G. West, *Multiple regression: Testin and interpreting interactions*. 1991.
 - [118] P. W. Holland, "Statistics and causal inference," *Journal of the American Statistical Association*, vol. 81, no. 396, pp. 945–960, 1986.
 - [119] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. 2010.
 - [120] K. V. Desouza, "Intrapreneurship - Managing ideas within your organization," *Technological Forecasting and Social Change*, vol. 91, pp. 352–353, 2014.
 - [121] S. Gupta, S. Bhardwaj, P. Dmitriev, U. Lucy, A. Fabijan, and P. Raff, "The Anatomy of a Large-Scale Online Experimentation Platform," in *to appear in Proceedings of the 2018 IEEE International Conference on Software Architecture (ICSA)*, 2018.
 - [122] A. Deng and X. Shi, "Data-Driven Metric Development for Online Controlled Experiments," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, 2016, pp. 77–86.
 - [123] J. Bosch, *Speed, Data, and Ecosystems: Excelling in a Software-Driven World*. CRC Press, 2017.
 - [124] V. R. Basili, R. W. Selby, and D. H. Hutchens, "Experimentation in software engineering," *IEEE Transactions on software engineering*, no. 7, pp. 733–743, 1986.
 - [125] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*, vol. 9783642290. 2012.
 - [126] N. Juristo and A. M. Moreno, "Basics of Software Engineering Experimentation," *Analysis*, vol. 5/6, p. 420, 2001.
 - [127] S. Blank, "Why the lean start-up changes everything," *Harvard Business Review*, vol. 91, no. 5. John Wiley & Sons, p. 288, 2013.
 - [128] G. Schermann, J. J. Cito, and P. Leitner, "Continuous Experimentation: Challenges, Implementation Techniques, and Current Research," *IEEE Software*, vol. 35, no. 2, pp. 26–31,

- Mar. 2018.
- [129] J. F. Box, “R.A. Fisher and the Design of Experiments, 1922–1926,” *The American Statistician*, vol. 34, no. 1, pp. 1–7, Feb. 1980.
 - [130] S. D. Simon, “Is the randomized clinical trial the gold standard of research?,” *Journal of Andrology*, vol. 22, no. 6, pp. 938–943, Nov. 2001.
 - [131] A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch, “Online Controlled Experimentation at Scale: An Empirical Survey on the Current State of A/B Testing,” in *Submission*, 2018.
 - [132] A. Fabijan, P. Dmitriev, H. H. Olsson, and J. Bosch, “The Evolution of Continuous Experimentation in Software Product Development,” in *Proceedings of the 39th International Conference on Software Engineering ICSE’17*, 2017.
 - [133] T. Fawcett, “Data Science and Its Relationship to Big Data and Data-Driven Decision Making Data Science and its relationship to Big Data and data-driven decision making,” no. April, 2017.
 - [134] S. Miller and D. Hughes, “The Quant Crunch: How the Demand For Data Science Skills is Disrupting the Job Market,” *Burning Glass Technologies*, 2017.
 - [135] R. L. Kaufman, J. Pitchforth, and L. Vermeer, “Democratizing online controlled experiments at Booking. com,” *arXiv preprint arXiv:1710.08217*, pp. 1–7, 2017.
 - [136] 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 ’14*, 2014, pp. 283–292.
 - [137] P. Runeson and M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2008.
 - [138] R. K. Yin, *Case study research: design and methods*, vol. 5., no. 5. 2009.
 - [139] K. M. Eisenhardt, “Building Theories from Case Study Research,” *Academy of Management Review*, vol. 14, no. 4, pp. 532–550, 1989.
 - [140] L. Aaboen, A. Dubois, and F. Lind, “Capturing processes in longitudinal multiple case studies,” *Industrial Marketing Management*, vol. 41, no. 2, pp. 235–246, 2012.
 - [141] A. B. Sandberg and I. Crnkovic, “Meeting industry-academia research collaboration challenges with agile methodologies,” *Proceedings - 2017 IEEE/ACM 39th International Conference*

- on Software Engineering: Software Engineering in Practice Track, ICSE-SEIP 2017, pp. 73–82, 2017.
- [142] A. B. Sandberg, “Agile Collaborative Collaboration,” *IEEE computer society*, vol. 28, no. 4, pp. 74–84, 2011.
 - [143] M. Crotty, *The Foundations of Social Research: Meaning and Perspective in the Research Process*. 1998.
 - [144] J. E. Guillory *et al.*, “Experimental evidence of massivescale emotional contagion through social networks,” *Proceedings of the National Academy of Sciences*, vol. 111, no. 29, pp. 10779–10779, Jul. 2014.
 - [145] J. M. Balkin, “Information Fiduciaries and the First Amendment,” *U.C. Davis Law Review*, vol. 49, no. 4, p. 1183, 2016.
 - [146] H. T. Tavani, “Ethics and technology: Ethical issues in an age of information and communication technology,” 2003.
 - [147] L. Atzori, A. Iera, and G. Morabito, “The internet of things: A survey,” *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.
 - [148] A. Fabijan, H. H. Olsson, and J. Bosch, “Customer Feedback and Data Collection Techniques in Software R{&}D: A Literature Review,” in *Software Business: 6th International Conference, ICSOB 2015, Braga, Portugal, June 10-12, 2015, Proceedings*, J. M. Fernandes, R. J. Machado, and K. Wnuk, Eds. Cham: Springer International Publishing, 2015, pp. 139–153.
 - [149] L. Williams and A. Cockburn, “Agile software development: It’s about feedback and change,” *Computer*, vol. 36, no. 6, p. 39, 2003.
 - [150] H. H. Olsson and J. Bosch, “Towards Data-Driven Product Development: A Multiple Case Study on Post-deployment Data Usage in Software-Intensive Embedded Systems,” in *Lean Enterprise Software and Systems - Proceedings of the 4th International Conference on Lean Enterprise Software and Systems, LESS 2013, Galway, Ireland, December 1-4, 2013*, 2014, pp. 152–164.
 - [151] G. Kotonya and I. Sommerville, “Requirements Engineering: Processes and Techniques (Worldwide Series in Computer Science),” *Star*, p. 294, 1998.
 - [152] S. E. Sampson, “Ramifications of Monitoring Service Quality Through Passively Solicited Customer Feedback,” *Decision Sciences*, vol. 27, no. 4, pp. 601–622, 1996.

- [153] J. Bosch, "Building products as innovation experiment systems," *Lecture Notes in Business Information Processing*, vol. 114 LNBIP, pp. 27–39, 2012.
- [154] R. Kohavi, R. Longbotham, D. Sommerfield, and R. M. Henne, "Controlled experiments on the web: survey and practical guide," *Data Mining and Knowledge Discovery*, vol. 18, no. 1, pp. 140–181, Feb. 2009.
- [155] H. Saarijärvi, H. Karjaluo, and H. Kuusela, "Customer relationship management: the evolving role of customer data," *Marketing intelligence & planning*, vol. 31, no. 6, pp. 584–600, 2013.
- [156] A. Fabijan, H. H. Olsson, and J. Bosch, "Commodity Eats Innovation for Breakfast: A Model for Differentiating Feature Realization," in *Proceedings of the Product-Focused Software Process Improvement: 17th International Conference, PROFES 2016, Trondheim, Norway, November 22-24, 2016, Proceedings*, 2016, pp. 517–525.
- [157] A. Fabijan, "Developing the right features: the role and impact of customer and product data in software product development," Malmö University, 2016.
- [158] S. Blank, "Why the Lean Start Up Changes Everything," *Harvard Business Review*, vol. 91, no. 5, p. 64, May 2013.
- [159] J. Bosch, "Achieving Simplicity with the Three-Layer Product Model," *Computer*, vol. 46, no. 11, pp. 34–39, Nov. 2013.
- [160] G. A. Moore, *Dealing with Darwin: How great companies innovate at every phase of their evolution*. Penguin, 2005.
- [161] P. Mayring, "Qualitative content analysis - research instrument or mode of interpretation," in *The Role of the Researcher in Qualitative Psychology*, 2002, pp. 139–148.
- [162] S. Augustine, B. Payne, F. Sencindiver, and S. Woodcock, "Agile Project Management: Steering from the Edges," *Commun. ACM*, vol. 48, no. 12, pp. 85–89, Dec. 2005.
- [163] A. Fabijan, H. H. Olsson, and J. Bosch, "Early Value Argumentation and Prediction: An Iterative Approach to Quantifying Feature Value," in *Proceedings of the 16th International Conference on Product-Focused Software Process Improvement (PROFES 2015), Bolzano, Italy, December 2-4, 2015*, vol. LNCS 9459, P. Abrahamsson, L. Corral, M. Oivo, and B. Russo, Eds. Cham: Springer International Publishing, 2015, pp. 16–23.
- [164] D. L. Parnas, "Software aging," *16th International Conference on Software Engineering*, pp. 279–287, 1994.

- [165] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *ACM SIGSOFT Software Engineering Notes*, vol. 17, no. 4, pp. 40–52, 1992.
- [166] M. Calder, M. Kolberg, E. H. Magill, and S. Reiff-Marganiec, "Feature interaction: A critical review and considered forecast," *Computer Networks*, vol. 41, no. 1. pp. 115–141, 2003.
- [167] J. M. Sullivan, "Impediments to and Incentives for Automation in the Air Force," in *Proceedings of ISTAS 2005, International Symposium on Technology and Society. Weapons and Wires: Prevention and Safety in a Time of Fear*, 2005, pp. 102–110.
- [168] D. A. Norman, *The Design of Everyday Things*, vol. 16, no. 4. 2002.
- [169] R. G. Cooper, S. J. Edgett, and E. J. Kleinschmidt, "New product portfolio management: practices and performance," *Journal of Product Innovation Management*, vol. 16, no. 4, pp. 333–351, 1999.
- [170] D. Leonard-Barton, "Core capabilities and core rigidities: A paradox in managing new product development," *Strategic Management Journal*, vol. 13, pp. 111–125, 1992.
- [171] L. Gorchels, *The Product Manager's Handbook*. McGraw Hill Professional, 2000.
- [172] D. Cohen, G. Larson, and B. Ware, "Improving software investments through requirements validation," *Proceedings 26th Annual NASA Goddard Software Engineering Workshop*, pp. 106–114, 2001.
- [173] A. F. Payne, K. Storbacka, and P. Frow, "Managing the co-creation of value," *Journal of the Academy of Marketing Science*, vol. 36, no. 1, pp. 83–96, 2008.
- [174] M. E. Porter, "Competitive Advantage," *Strategic Management*, vol. May-June. pp. 1–580, 1985.
- [175] J. Hemilä, J. Vilko, E. Kallionpää, and J. Rantala, "Value creation in product-service supply networks," in *The Proceedings of 19th International Symposium on Logistics (ISL 2014)*, 6.-9.7.2014., Ho Chi Minh City, Vietnam, Nottingham University Business School, 2014, pp. 168–175.
- [176] R. C. Van Nostrand, "Design of Experiments Using the Taguchi Approach: 16 Steps to Product and Process Improvement," *Technometrics*, vol. 44, no. 3, pp. 289–289, Aug. 2002.
- [177] R. S. Kaplan and D. P. Norton, "The Balanced Scorecard:

- Translating Strategy Into Action,” *Harvard Business School Press*, pp. 1–311, 1996.
- [178] J. S. Pinegar, “What Customers Want: Using Outcome-Driven Innovation to Create Breakthrough Products and Services by Anthony W. Ulwick,” *Journal of Product Innovation Management*, vol. 23, no. 5, pp. 464–466, 2006.
 - [179] V. Basili, J. Heidrich, M. Lindvall, J. Münch, M. Regardie, and A. Trendowicz, “GQM+Strategies - Aligning business strategies with software measurement,” in *Proceedings - 1st International Symposium on Empirical Software Engineering and Measurement, ESEM 2007*, 2007, pp. 488–490.
 - [180] H.-F. Hsieh and S. E. Shannon, “Three approaches to qualitative content analysis,” *Qualitative health research*, vol. 15, no. 9, pp. 1277–88, Nov. 2005.
 - [181] M. Jackson and M. Pamela Zave, “Distributed feature composition: A virtual architecture for telecommunications services,” *IEEE Transactions on Software Engineering*, vol. 24, no. 10, pp. 831–847, 1998.
 - [182] J. Cleland-Huang, “Stakeholders on the Prowl,” *IEEE Software*, vol. 33, no. 2, pp. 29–31, Mar. 2016.
 - [183] K. Rodden, H. Hutchinson, and X. Fu, “Measuring the User Experience on a Large Scale: User-Centered Metrics for Web Applications,” *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 2395–2398, 2010.
 - [184] H. Hohnhold, D. O’Brien, and D. Tang, “Focusing on the Long-term,” in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD ’15*, 2015, pp. 1849–1858.
 - [185] “Hypothesis Kit for A/B testing.” [Online]. Available: <http://www.experimentationhub.com/hypothesis-kit.html>.
 - [186] J. Reason, “Human error: models and management,” *Bmj*, vol. 320, no. March, pp. 768–770, Mar. 2000.
 - [187] T. Crook, B. Frasca, R. Kohavi, and R. Longbotham, “Seven pitfalls to avoid when running controlled experiments on the web,” in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD ’09*, 2009, p. 1105.
 - [188] A. Deng, Y. Xu, R. Kohavi, and T. Walker, “Improving the sensitivity of online controlled experiments by utilizing pre-experiment data,” in *Proceedings of the sixth ACM international conference on Web search and data mining - WSDM ’13*, 2013, p. 10.

- [189] T. Kluck and L. Vermeer, "Leaky Abstraction In Online Experimentation Platforms: A Conceptual Framework To Categorize Common Challenges," Oct. 2017.
- [190] B. Long and B. Chen, "An Empirical Study on Recommendation with Multiple Types of Feedback," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining(KDD 2016)*, pp. 283–292, 2016.
- [191] W. Maalej, H.-J. Happel, and A. Rashid, "When Users Become Collaborators: Towards Continuous and Context-Aware User Input," in *OOPSLA '09 Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, 2009, pp. 981–990.
- [192] R. Kohavi, A. Deng, R. Longbotham, and Y. Xu, "Seven rules of thumb for web site experimenters," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14*, 2014, pp. 1857–1866.
- [193] J.-C. Plantin, C. Lagoze, P. N. Edwards, and C. Sandvig, "Infrastructure studies meet platform studies in the age of Google and Facebook," *New Media & Society*, pp. 1–18, 2016.
- [194] J. A. Obar, "The Biggest Lie on the Internet: Ignoring the Privacy Policies and Terms of Service Policies of Social Networking Services," *SSRN Electronic Journal*, vol. 15, no. 2, pp. 67–82, 2016.
- [195] D. Yuan, S. Park, and Y. Zhou, "Characterizing logging practices in open-source software," in *Proceedings - International Conference on Software Engineering*, 2012, pp. 102–112.
- [196] A. Katal, M. Wazid, and R. H. Goudar, "Big data: Issues, challenges, tools and Good practices," *2013 6th International Conference on Contemporary Computing, IC3 2013*, pp. 404–409, 2013.
- [197] E. Johansson, D. Bergdahl, J. Bosch, and H. H. Olsson, "Requirement Prioritization with Quantitative Data - a case study," in *International Conference on Product-Focused Software Process Improvement*, 2015, vol. 9459, pp. 89–104.
- [198] D. Yuan, J. Zheng, S. Park, Y. Zhou, and S. Savage, "Improving software diagnosability via log enhancement," *ACM SIGPLAN Notices*, vol. 47, no. 4, p. 3, 2012.

- [199] G. Keppel, W. H. Saufley, and H. Tokunaga, *Introduction to design and analysis: A student's handbook*. Macmillan, 1992.
- [200] D. J. Patil, "Building Data Science Teams," *Oreilly Radar*, pp. 1–25, 2011.
- [201] D. S. Cruzes and T. Dyba, "Recommended Steps for Thematic Synthesis in Software Engineering," *2011 International Symposium on Empirical Software Engineering and Measurement*, no. 7491, pp. 275–284, 2011.
- [202] A. Noulas, "User Engagement through Topic Modelling in Travel," in *Proceeding of the Second Workshop on User Engagement Optimization*. 2014., 2014, no. 2014, pp. 2–7.
- [203] N. Tintarev and J. Masthoff, "Evaluating the effectiveness of explanations for recommender systems," *User Modeling and User-Adapted Interaction*, vol. 22, no. 4, pp. 399–439, Oct. 2012.
- [204] Nasdaq, "Intuit Q3 2017 Earnings Conference Call," *Nasdaq*, 2017. [Online]. Available: <http://www.nasdaq.com/aspx/call-transcript.aspx?StoryId=4075935&Title=intuit-s-intu-ceo-brad-smith-on-q3-2017-results-earnings-call-transcript>.
- [205] R. B. Bausell and Y.-F. Li, *Power analysis for experimental research: a practical guide for the biological, medical and social sciences*. Cambridge University Press, 2002.
- [206] R. Higham, "Experimentation Hub," 2016. [Online]. Available: <http://www.experimentationhub.com/>.
- [207] A. Edmondson, "Psychological Safety and Learning Behavior in Work Teams," *Administrative Science Quarterly*, vol. 44, no. 2, p. 350, 1999.
- [208] R. Al-Lamee *et al.*, "Percutaneous coronary intervention in stable angina (ORBITA): a double-blind, randomised controlled trial," *The Lancet*, Nov. 2017.
- [209] L. Delizonna, "High-Performing Teams Need Psychological Safety. Here 's How to Create It," *Harvard Business Review*, no. August, pp. 2–6, 2017.
- [210] A. Chaudhuri, "Internet of things data protection and privacy in the era of the General Data Protection Regulation," *Journal of Data Protection & Privacy*, vol. 1, no. 1, pp. 64–75, 2016.
- [211] E. Bjarnason and M. Borg, "Aligning Requirements and Testing: Working Together toward the Same Goal," *IEEE Software*, vol. 34, no. 1, pp. 20–23, Jan. 2017.
- [212] O. J. Dunn, "Multiple Comparisons among Means," *Journal of the American Statistical Association*, vol. 56, no. 293, pp. 52–64, 1961.

- [213] R. Conradi and B. Westfechtel, "Version Models for Software Configuration Management," *ACM Comput. Surv.*, vol. 30, no. 2, pp. 232–282, Jun. 1998.
- [214] P. Dmitriev and A. Fabijan, "Experimentation Growth," 2017. [Online]. Available: <https://www.exp-growth.com>.
- [215] M. Kumar, M. Shukla, and S. Agarwal, "A Hybrid Approach of Requirement Engineering in Agile Software Development," *2013 International Conference on Machine Intelligence and Research Advancement*, pp. 515–519, 2013.
- [216] V. R. Basili, G. Caldiera, and H. D. Rombach, "Experience factory," *Encyclopedia of software engineering*, vol. 1, pp. 469–476, 1994.
- [217] ExP, "Microsoft Experimentation Platform." [Online]. Available: <http://www.exp-platform.com>.
- [218] A. Deng, J. Lu, and S. Chen, "Continuous monitoring of A/B tests without pain: Optional stopping in Bayesian testing," in *Proceedings - 3rd IEEE International Conference on Data Science and Advanced Analytics, DSAA 2016*, 2016, pp. 243–252.
- [219] R. Kohavi and R. Longbotham, "Online experiments: Lessons learned," *Computer*, vol. 40, no. 9, pp. 103–105, 2007.
- [220] R. Kohavi, A. Deng, B. Frasca, R. Longbotham, T. Walker, and Y. Xu, "Trustworthy Online Controlled Experiments : Five Puzzling Outcomes Explained," *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 786–794, 2012.
- [221] K. L. Morgan and D. B. Rubin, "Rerandomization to improve covariate balance in experiments," *Annals of Statistics*, vol. 40, no. 2, pp. 1263–1282, 2012.
- [222] F. Radlinski and N. Craswell, "Optimized interleaving for online retrieval evaluation," in *Proceedings of the sixth ACM international conference on Web search and data mining - WSDM '13*, 2013, p. 245.
- [223] A. Deng, P. Zhang, S. Chen, D. W. Kim, and J. Lu, "Concise Summarization of Heterogeneous Treatment Effect Using Total Variation Regularized Regression," *In submission*, Oct. 2016.
- [224] "Multiple Comparisons Problem," *Wikipedia*. .
- [225] P. K. Pearson, "Fast hashing of variable-length text strings," *Communications of the ACM*, vol. 33, no. 6, pp. 677–680, Jun. 1990.
- [226] F. A. Manuele, "Leading & Lagging Indicators," *Professional*

- Safety*, vol. 54, no. 12, pp. 28–33, 2009.
- [227] R. K. Roy, *Design of Experiments Using The Taguchi Approach: 16 Steps to Product and Process Improvement*. 2001.
 - [228] Z. Zhao, M. Chen, D. Matheson, and M. Stone, “Online Experimentation Diagnosis and Troubleshooting Beyond AA Validation,” in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, 2016, no. October 2016, pp. 498–507.
 - [229] A. Fabijan, P. Dmitriev, C. McFarland, L. Vermeer, H. H. Olsson, and J. Bosch, “The Experimentation Growth: Evolving Trustworthy A/B Testing Capabilities in Online Software Companies,” *In Revision in Journal of Software: Evolution and Process*, 2018.
 - [230] L. Madeyski and M. Kawalerowicz, “Software Engineering Needs Agile Experimentation: A New Practice and Supporting Tool,” in *Advances in Intelligent Systems and Computing*, vol. 504, L. Madeyski, M. Śmiałek, B. Hnatkowska, and Z. Huzar, Eds. Cham: Springer International Publishing, 2017, pp. 149–162.
 - [231] F. Girardin and N. Lathia, “When User Experience Designers Partner with Data Scientists,” in *AAAI 2017 SPRING SYMPOSIA*, 2017, pp. 376–381.
 - [232] V. F. Ridgway, “Dysfunctional consequences of performance measurements,” *Administrative Science Quarterly*, vol. 1, no. 2, pp. 240–247, 1956.
 - [233] A. Fabijan, H. H. Olsson, and J. Bosch, “Time to Say ‘Good Bye’: Feature Lifecycle,” in *Proceedings of the 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Limassol, Cyprus. 31 Aug.-2 Sept. 2016, 2016, pp. 9–16.
 - [234] A. M. Ghalayini and J. S. Noble, “The changing basis of performance measurement,” *International Journal of Operations & Production Management*, vol. 16, no. 8, pp. 63–80, Aug. 1996.
 - [235] C. McChesney, S. Covey, and J. Huling, “The 4 disciplines of execution: achieving your wildly important goals,” *Four disciplines of execution*, pp. 1–8, 2012.
 - [236] E. A. Locke, K. N. Shaw, L. M. Saari, and G. P. Latham, “Goal setting and task performance: 1969-1980,” *Psychological Bulletin*, vol. 90, no. 1, pp. 125–152, 1981.
 - [237] X. M. Bezuijen, K. van Dam, P. T. van den Berg, and H.

- Thierry, “How leaders stimulate employee learning: A leader-member exchange approach,” *Journal of Occupational & Organizational Psychology*, vol. 83, no. 3, pp. 673–693, 2010.
- [238] E. a Locke and G. P. Latham, “Building a practically useful theory of goal setting and task motivation: A 35-year odyssey,” *The American Psychologist*, vol. 57, no. 9, pp. 705–717, 2002.
- [239] C. Robson, *Real World Research: A Resource for Social Scientists and Practitioner-Researchers*. 2002.
- [240] K.-D. Althoff, B. Decker, S. Hartkopf, A. Jedlitschka, M. Nick, and J. Rech, “Experience Management: The Fraunhofer IESE Experience Factory,” in *Industrial Conference on Data Mining*, 2001, pp. 12–29.
- [241] D. I. Mattos, J. Bosch, and H. H. Olsson, “Challenges and Strategies for Undertaking Continuous Experimentation to Embedded Systems: Industry and Research Perspectives,” in *19th International Conference on Agile Software Development, XP’18*, 2018, no. March, pp. 1–15.

ISBN 978-91-7104-918-6 (print)

ISBN 978-91-7104-919-3 (pdf)

MALMÖ UNIVERSITY
205 06 MALMÖ, SWEDEN
WWW.MAU.SE