Experiment-Driven Product Development

How to Use a Data-Informed Approach to Learn, Iterate, and Succeed Faster

Paul Rissen

Apress®

EXPERIMENT-DRIVEN PRODUCT DEVELOPMENT

HOW TO USE A DATA-INFORMED APPROACH TO LEARN, ITERATE, AND SUCCEED FASTER

Paul Rissen

Apress®

Experiment-Driven Product Development: How to Use a Data-Informed Approach to Learn, Iterate, and Succeed Faster

Paul Rissen
Middlesex, UK

ISBN-13 (pbk): 978-1-4842-5527-8 ISBN-13 (electronic): 978-1-4842-5528-5
https://doi.org/10.1007/978-1-4842-5528-5

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Shiva Ramachandran
Development Editor: Laura Berendson
Coordinating Editor: Rita Fernando

Cover designed by eStudioCalamar

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit http://www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-5527-8. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

To my parents, my brother, my colleagues past and present—you all gave me the courage and skills to get me to where I am today. Thank you.

Contents

About the Author

Paul Rissen has worked in digital product development for 13 years, starting as an information and UX architect at Siemens. He worked with the BBC for almost 10 years, covering the development of the first version of iPlayer and various education and news products. Stepping into the role of product manager, first at the BBC and then with Springer Nature, Paul collaborated with others to develop the experiment-driven approach, seeking to share this knowledge by writing this book.

Acknowledgments

First and foremost, this book would not have been possible without the support of my parents and brother. Thank you for all your support.

Secondly, Todd Carter, Caitlin Loftus, and the rest of Team Recommended—Dan, Eugene, Andy, Laura, Sarah, Gordon, and Gavin. Thank you for being part of an amazing team. This book is dedicated to you all, so that what we developed lives on.

Thank you, too, Marta Rolak DeLorca and Jonathan Austin for giving me the opportunity to work with such great colleagues.

To all my friends, colleagues, and mentors from the past—Tom Scott, Michael Smethurst, and Chris Sizemore. To Mike Atherton and Carrie Hane, whose fault this book is, and to Jo Weber, who gave me the shot of confidence I needed, all those years ago in youth work.

Introduction

This is a book about experimentation. It focuses on the process of research, discovery, and learning, as applied to digital product design and development. Using the concept of the experiment, this book sets out a framework for asking questions and discovering answers in a structured form.

Rather than treating research as a separate task from interface design and the writing of code, this book approaches the *entire practice* of product development as one of research. In experiment-driven product development (XDPD), **discovery is a mindset, not a project phase.** Everyone on a product team, every activity, can be part of the process of research, discovery, and learning.

Questions, not solutions, are the epicenter of this approach. Instead of generating ideas and seeking to validate them, XDPD places the emphasis on uncovering assumptions, generating questions, and seeking to find useful, reliable answers.

XDPD won't tell you what product, service, or feature to design or build. It will help you design and build better products, services, and features, no matter their form or function.

With all this in mind, then, let's dive in straight away by designing our first experiment.

Our first experiment

Premise: Reading the introduction to this book will help me understand what this book is all about.

Question: What is "experiment-driven product development," and what will I learn from this book?

Experiment: Reading the first chapter of *Experiment-Driven Product Development*

Hypothesis: Given that I believe reading this book will help me in some way, I believe that by reading the first chapter of *Experiment-Driven Product Development*, I'll be able to explain to others what is meant by the term and what each chapter in the book will cover.

With the basics in place, let's begin the experiment….

Why write this book?

There are hundreds, if not thousands, of books on the market intended to help people working in digital industries craft better products. Why add another to the shelves?

The answer is that this book is designed to be a practical guide to an often overlooked part of product design and development—the craft of discovery, research, and learning. Plenty of good words have been written and said about how to rapidly come up with, design, and validate ideas, but less has been written about what comes before the ideas—the questions—and, crucially, how you can discover the answers to those questions in a useful, reliable way.

Experiment-driven product development is not intended as a replacement for, or alternative to, agile or lean methodologies. It is a complement to them—a way of bringing rigor to the process of discovering potential product, service, or feature ideas and of framing user research, data analysis, and prototype development toward the achievement of team-defined goals. Finally, it provides a way of measuring, in as objective a way as possible, the success or otherwise of product ideas.

I've written this book, because, after almost three years of developing this practice through real-world product development, I believe it's simply a better, more interesting, creative, and efficient way of working.

This isn't to say that you should throw away all other product development methodologies and force everyone to follow the "one true path" of experimentation. Rather, I hope that this book gives the reader a greater appreciation of the craft of experimentation and allows you to take this approach, use it in your own context, and adapt where necessary.

Although this book is written to give you a clear understanding of the approach, as always, the only way to really learn is to experiment with it! You might find, as I have done, when bringing the approach to new teams, that it takes a few tries at it, before it feels comfortable. You might also find that not every single part of what I discuss in this book is applicable to your context. It might even be the case that you take what I recommend here and adapt it to your own situation.

That's fine—and good. Although I decided to sit down and write this book, I didn't develop the approach all on my own. Rather, it was shaped by three years of working with different teams, trying parts of the process, discussing honestly among ourselves, and refining as we went along. No one involved began with a fully-fledged version of this approach in mind. It is something that has, and hopefully will continue to, evolve over time.

I first joined a team working in this way in 2016. Over the course of the next 12 months, after intense discussion and development of the process,

we realized we were on to something. With the team dispersing, we realized that we hadn't captured the knowledge of this process in a single place nor yet made a real effort to share this framework with other teams.

Two members of the team put together an internal presentation explaining a version of the process, but aside from that, when the year ended, this way of working was at risk of being lost forever.[1]

This book is an attempt to capture three years' worth of honest discussion and development with multiple teams, so that others can take what we've done and run with it.

It will take you through the process of turning ideas, hunches, and requirements into premises and hypotheses. It uses the idea of an "experiment card" to walk through the practical steps of designing an experiment, measuring its success (or failure), presenting back to stakeholders, and where to go next.

I found experiment-driven product development to be an interesting, and very useful, way of approaching aspects of working on digital products and services, and it is my hope that it proves to be the same for you too.

How to use this book

This book is designed to be a practical guide to implementing an experiment-driven product development approach. That said, some context is required, and so the book is divided into three distinct parts.

Part 1: Introduction

Part 1 sets the scene and provides context. Its aim is to inspire you, by exploring the key concepts behind the approach.

Chapter 1 provides greater detail on what is, and isn't, meant by "experiment-driven product development," gives some reasons as to how the approach can benefit your team, and discusses the opportunities and challenges you might encounter when trying to implement this at the scale of a multi-team organization.

Chapter 2 looks at the different activities often involved in product development and explores how adopting an experiment-driven approach to each of them can help focus your work. Crucially, it seeks to show how, although experiments have traditionally been used to iterate on established product features, through A/B testing and the like, the concept of an experiment can be used in other ways to frame new feature development, qualitative research, and data analysis.

[1] It's worth noting that the title of that presentation was "*Being Experiment Driven is hard*"…

Chapter 3 lays out some of the key principles and tenets which underpin the experiment-driven approach. These lay out the philosophical foundations for any team adopting experiment-driven product development, and serve as a guide as to what to keep front of mind as you work together through this approach on a daily, weekly, and monthly basis.

Chapter 4, the final chapter in this part, expands upon Chapter 3 by exploring one of these key principles—that of the need to concentrate on the "Simplest Useful Thing." A proposed reframing of the "Minimum Viable Product" concept, a "Simplest Useful Thing" mindset, concentrates a team on a culture of learning as fast as can be. This chapter explores what "Simplest Useful Thing" means from the point of view of product development, designing experiments, and a technical team approach.

Part 2: Experiment-driven development in practice

Part 2 is the core of the book. Here, we focus on the practical elements of the experiment-driven product development approach. I introduce the "*Experiment Card*" artifact and use that as a way of exploring the different aspects of designing experiments that need to be considered.

Chapter 5 covers how you get started. It describes an approach for coming up with ideas for experiments, developing them into experiment-ready questions, and prioritizing your work.

Chapter 6 focuses on the things that everyone in a team needs to be clear on, before an experiment is run. It offers guidance on how to develop a precise, testable hypothesis; how to choose metrics which enable you to determine the results of an experiment; and which details you need to capture to ensure clarity among the team.

Chapter 7 considers the importance of designing your experiments so that you can draw meaningful, reliable conclusions from them. It looks at some statistical concepts and methods you can use to determine how much data you need to collect, how long to run an experiment, and how to deal with certainty and uncertainty in results.

Part 3: Aftermath of experiments

Part 3 is the final section of the book. Here, we focus on what happens *after* an experiment has been run.

Chapter 8 takes you through the things to look for once an experiment has concluded, in order to learn from it. An experiment really isn't worth running unless you take the time to analyze the results. It also begins looking to the future—how do you take what you've learned and begin the cycle over again.

Finally, **Chapter 9** takes you through how you can go about sharing the knowledge of what you learn from running experiments, with your stakeholders and other teams. We'll cover a share-back format which summarizes the "Experiment Card" concept but also links what a team has been working on, back to its overall goals.

And that, in summary, is what you can expect from reading this book—I hope it proves to be useful and interesting! So, with that out the way, let's look back at our first experiment and see what the result of running it was.

Summary

Premise: Reading the introduction to this book will help me understand what this book is all about.

Question: What is "experiment-driven product development," and what will I learn from this book?

Experiment: Reading the first chapter of *Experiment-Driven Product Development*.

Hypothesis: Given that I believe reading this book will help me in some way, I believe that by reading the first chapter of *Experiment-Driven Product Development*, I'll be able to explain to others what is meant by the term and what each chapter in the book will cover.

Results: Well, I'd need to gather the actual data from everyone reading this, of course, but I hope it has been a success.

By now, it should be clear that experiment-driven product development involves far more than getting a team to run some A/B tests. It's a whole approach to working out what you want to investigate, build, research, and test and a way to design these activities so that they can be as useful as possible.

Importantly, this approach isn't just about building things you already want to build, or mindless "innovation" projects. It's about learning, and failing fast—testing your assumptions, and getting to a better place, as quickly as possible.

Hopefully the hypothesis I suggested at the very beginning has been proven true—by reading this chapter, you now have a greater understanding of the purpose of the book, why you might want to read it, what the book covers, and where such a practice might fit in your work life.

With no further ado, therefore, let's move on to consider in greater detail what is meant by "experiment-driven product development" and start assessing the wider context within which this approach fits.

PART

1

Introduction

CHAPTER

1

What Is Experiment-Driven Product Development?

Experiment-driven product development (XDPD) is a way of approaching the product design and development process so that research, discovery, and learning—asking questions and getting useful, reliable answers—are prioritized over designing, and then validating, solutions.

In the XDPD framework, we use the concept of an experiment to frame almost everything the team does. All kinds of activities, from data analysis, through user research, to writing software, are seen through the lens of research. In this chapter, we'll cover

- What's involved in the process
- What we mean by "an experiment"
- How XDPD relates to the "Lean" approach

P. Rissen, *Experiment-Driven Product Development*,
https://doi.org/10.1007/978-1-4842-5528-5_1

- Why you should consider trying the XDPD approach
- How to get the best out of XDPD in a multi-team environment

How does XDPD work?

Asking questions, and using structured methods and tools to answer them, is not a revolutionary idea in and of itself. Scientists and researchers have been using what's known as "the scientific method" for centuries, as a way of seeking knowledge. What is at least *slightly* novel, however, is the application of this approach to encompass the majority of activities a modern digital product design and development team engages in.

Let's take a look at the basic process that a team using the XDPD framework would go through (Figure 1-1).

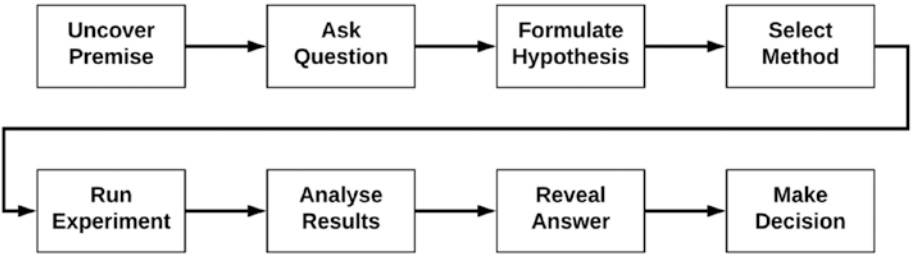

Figure 1-1. The basic process of experiment-driven product development

We kick off by uncovering premises. These premises can be based on prior knowledge, assumptions, feature ideas, or claims being made about the product or its users. These provide our "fuel" for experiments, and the next step is to take this fuel and turn premises into questions.

Questions form the absolute bedrock of the XDPD approach, and pretty much everything else stems from them. Once we have a question we want to answer, we can formulate a hypothesis—a statement that sums up what we believe the answer might be—and proceed to design our experiment in a way which will test this hypothesis.

We choose a method, or form in which the experiment will manifest, run the experiment, and then dig into the results. By analyzing the results, we should uncover an answer, along with other interesting pieces of knowledge that can inform future experiments. Finally, the answer that is revealed should help us make a decision. This could be a decision around what to do next, whether to invest more time in something or to abandon this line of inquiry.

Parts 2 and 3 of this book will go into much more detail around each of the stages outlined earlier, but now you should have a fair idea of the basic outline of how this will work. Perhaps this is completely new to you, perhaps not. The crucial thing to consider here is how you might apply this to the different activities that a team takes part in. That brings us on to the next question—what *do* we mean by "an experiment," anyway?

What do we mean by "an experiment"?

What do you think of when you hear the term "experiment"? It's likely to be one of two things.

Misconception #1: Experiment = A/B test

Firstly, most people in the realm of technology think of A/B tests—where one set of users is exposed to the existing, "control" version of a product and another set of users is exposed to a slightly different, experimental version of the product. Their reactions are measured, usually according to some predefined success criteria; the two results compared, and a winner emerges.

There are more advanced forms of this kind of experimentation. Multivariate testing doesn't just give users exposure to one "new" version and the existing version of a product—it exposes users to several different variations, all competing with each other. At the extreme end, this can result in the infamous example of Google testing 41 different shades of blue—a great example of how not to apply an experiment-driven approach.[1]

Some teams even incorporate an aspect of machine learning into multivariant testing, using a technique called *multiarmed bandits*. In this format, several different options are tried, and after a while, the "worst" solution is dropped, while the experiment continues until there is a clear winner.

I'd be lying if I said that this book wasn't going to cover, or indeed take many principles from, the concepts behind A/B and multivariate testing. Indeed, when I first joined the team that helped put together the experiment-driven approach, every experiment was essentially a simple A/B test.

Over time, however, we realized that not only did we have a lot to learn about how to run A/B, let alone multivariate, tests properly, but that more importantly, the idea of an experiment was far more useful than our constrained definition.

[1] https://iterativepath.wordpress.com/2012/10/29/testing-40-shades-of-blue-ab-testing/

What we came to realize is crucial to this whole book. Once we understood the importance of designing the experiment we were planning to run, we began to see that deciding the method up front—regardless of whether it was an A/B test, a piece of user research, or something else entirely—was a mistake.

It didn't matter what exactly we were going to do—we needed to clarify the same points, over and over again:

- What question are we trying to answer?
- Why is it important to us?
- What do we think the answer might be?
- How much evidence will we need to gather in order to get a useful, reliable answer?
- What are we going to do to answer the question?
- What was the answer?
- What should we conclude from this?

Experiments thus became a structured way of asking, and answering, questions. They are a way of constructing a question which helps guide what you can do to discover an answer and a set of guidelines which you can use to ensure that the answer is reliable and useful.

Most importantly, any activity undertaken by the team could be in service to answering the question. It's not a question of having your UX team members run research sessions, while your developers are building software. User research is a way of answering a question. Running an A/B test is, too. Releasing a feature out into the world, no matter its form—a paper prototype or live code—should be framed around asking and answering the questions that will help you progress and build a better product.[2]

Remember! Not every experiment has to be an A/B test. Think of experiments as a structured way of asking questions. The exact method you use to answer the question can differ from experiment to experiment.

This is also nothing new. It's a way of thinking that has served scientists, among others, pretty well for the past 300 or so years. That leads us on to the second thing that might come to mind when you first hear the term "experiment."

---

[2] Arguably, refactoring and reducing technical or user experience debt is the exception to the rule.

Misconception #2: Experiments don't need purpose—they're "innovation"

"Experiment" brings to mind the image of the mad scientist, or perhaps in the context of digital product development, the research and development team. When companies want to get ahead of the curve, they often spin up a separate "innovation" team, away from the day-to-day work of running the business, as a way to scout out opportunities for future growth.

This itself is not a bad thing—provided that the innovation is guided in some way. It's not enough to tell a team to start innovating or experimenting. Experimenting for the sake of experimenting—running experiments in a haphazard way, with no particular purpose or direction, will not get you far.

Instead, experiments need to be driven by a sense of why they are being run in the first place. The emphasis in the long run shouldn't necessarily be on the "experiment" at all—that's just a means to an end. The key thing is the question, and answering it.

Ultimately, there needs to be a reason, a justification, for running an experiment—something you want to know the answer to. The beauty of the XDPD approach is that by framing product development in terms of research, discovery, and learning, teams are free to do anything and everything to answer the questions. They're not held to specific features or deliverables. The most important thing is to gain knowledge which will ultimately help you achieve a wider objective.

How does the XDPD approach relate to "Lean"?

Starting with Eric Ries' *The Lean Startup* and evolving through books such as Jeff Gothelf and Josh Seden's *Lean UX*, the last ten years or so have seen a massive shift in the way we think about software, product, and user experience development.[3,4] The "Lean" approach has become the go-to standard for any organization wanting to rapidly develop products at relatively low risk.

Experiment-driven product development comes from the same philosophical school of thought as Lean, and shares many similar principles, as you'll discover in this book. The importance of discovery, of doing the simplest thing in order to learn, and seeking to test assumptions, will be familiar to anyone who has read from the Lean canon.

---

[3] *The Lean Startup*, Eric Ries, Crown Publishing Group, 2011
[4] *Lean UX* (2nd Edition), Jeff Gothelf & Josh Seiden, O'Reilly Media, 2016

XDPD is an evolution of the Lean ethos rather than a totally different approach. In some ways, it's one of many possible iterations upon it. XDPD takes the idea of a hypothesis-driven, experimental approach to product development and goes both wider and deeper. By shifting the focus from assumptions to the step beforehand—questions—and going into more depth on how to design effective experiments, XDPD seeks to reinforce the basic tenets of Lean while bringing rigor and a different perspective to the approach.

Why should I consider trying this approach?

By this point, you're probably thinking, "oh no, yet another methodology for lean/agile digital product development." So why do I believe that it's worth being open to learning about this approach, and considering how you might apply it in your day-to-day practice? Let's list out the reasons, then I'll go through them, one by one.

Why experiment-driven product development? Because

- It forces you to challenge the premises underlying your work
- It lowers the cost of failure
- It helps you uncover new ideas through focusing on questions
- It enables you to treat your users as an equal stakeholder
- It helps you focus on things that actually make a difference
- It's fun!

It forces you to challenge the premises underlying your work

Firstly, as you'll see in the course of this book, most experiments are formed around premises—beliefs that you have around what might, and might not, be successful. Now, of course, some premises are helpful. I'm not suggesting you spend weeks and months testing the basic principles of web design or user experience.

However, some premises can also be dangerous—particularly when it comes to assumed knowledge about the audience, or assumed knowledge about what the "right" or "best" way of improving a product, or reaching your targets, might be. Some of those premises, most likely based on experience, may

well be correct—but without hard evidence to back them up, there's no way of knowing whether it was your knowledge, experience, wisdom, and insight which made the product successful, or blind luck.

Your objective, as a product development team, should be to always be learning—learning about what actually works and, just as, if not more, importantly, learning when you are *wrong*. Experiments are a great way of challenging the premises that underlie your work and gathering proof—to a reasonable degree of confidence—as to their validity or otherwise.

Lowering the cost of failure

And if the premise you were working with turns out to be misguided? Well, that's good! Better to know earlier, with a small but meaningful change made to the product, than to spend weeks and months developing something that you think is going to be a killer feature and turns out to just scupper your chances of surviving the next budgeting round.

Adopting an experiment-driven approach means being humble—being prepared to accept the evidence that your premises were wrong—but it also means that in doing so, you reduce the cost of failure. You may fail; you may make mistakes, make bets on ideas for features that may not earn you adoration from your users, but at least you'll do so in a way that minimizes the risk to the overall health of your product.

The sooner you know that something is wrong, with as small an intervention as possible, the quicker you can put it right and learn what not to do again. This means that as a team, you have to accept that you will make mistakes and that some of your ideas will be wrong. But that doesn't mean that you, or your team, are a failure—as long as you've gathered evidence and learned—you've actually increased your chances of success in the long term.

■ **Remember!** Discovering that a premise you were working upon was wrong is a good thing—but it's important to test these premises cheaply, so that if you are wrong, the impact on users is smaller.

Uncover new ideas through focusing on questions

Indeed, this can lead to one of the pleasantly surprising aspects of experiment-driven product development. Every experiment should revolve around a question that you want to answer. Sometimes, this will be a case of questioning a previous belief. Sometimes, though, an experiment can start from the opposite—a distinct lack of knowledge about something.

In answering those questions through experiments, this can inspire new breakthroughs, new ideas, and new innovations, which otherwise would only be able to emerge through potentially misleading premises and prior assumptions. ***Questions are the key to the XDPD approach.***

Treat your users as an equal stakeholder

Business stakeholders. Love 'em or hate 'em, they are the people you work with day in, day out. Despite most people's best intentions to put the user first, often, when push comes to shove, it's the people you work with every day who tend have the loudest voice on what happens with your product. Experiment-driven product development gives users a seat at the table, and a powerful voice when it comes to decisions, by giving you the ability to treat data—evidence of how they use your product and how it fits into the context of their lives- as a stakeholder.

More specifically, running experiments, and gathering evidence of what your users do in certain circumstances, enables you to make more informed decisions over whether to move forward with an idea.

Evidence of in situ reactions of users, as they go about their daily lives, gives you a set of facts upon which you can make a decision, taking you out of the realm of the mythological product manager who uses pure intuition, experience, and, frankly, clairvoyance, to make the "right" choice.

If the various opinions or political seniority of your business stakeholders is all you have to base your decisions on, then the user is often squeezed out. Whereas if you can point to evidence that users in the real world just don't behave in the way someone assumes, discussions move from an argument over opinions to action, grounded in facts.

It's important, of course, that this evidence is gathered under the right conditions—you want to ensure, as far as possible, that the decisions you make are based on evidence which is representative and reliable. That's where this book can help you.

Remember! Until you've tried something via a well-designed experiment, you just don't know whose voice to listen to. Use experiments as a way to gather evidence directly from your users and make a data-informed decision about how to proceed.

This is where the process of designing your experiments comes in—fortunately, this is what lies at the heart of this book. By carefully designing your experiments, you inevitably end up listening to your users at the right scale to make an informed decision, giving your users a representative voice in decision-making.

No matter what method you end up using in an experiment, taking into account the scale needed to achieve a reliable, useful answer is crucial. In doing so, you allow the actual behavior, thoughts, and feelings of your users, to influence things just as much as the people higher up the organization chart.

Focus on things that make a difference

Designing experiments forces you to think about what change you would like to see in the world. It gives form and structure to your hunches about what might be a good idea to reach a particular target, or achieve a particular objective.

The benefit of adopting an experiment-driven product development approach is that almost every experiment should clearly state what kind of change or difference you expect to see as a result.

In an experiment, you define a hypothesis—by doing "X," we believe we will see result "Y"—and then measure a number of key metrics, in order to determine whether your hypothesis is indeed correct.

Using this approach as much as possible in the product development context means that you reduce the amount of effort spent on work that doesn't relate directly to what you're ultimately trying to achieve.

If someone is lobbying for your team to spend time on their idea, and, through creating a hypothesis as part of the experiment design phase, it's proving impossible to describe how implementing that idea might lead to a change or difference in your key success metrics, then arguably you shouldn't be prioritizing that work.

It's fun!

Hopefully by now I've managed to convince you that experiment-driven product development is something worth exploring. But there's one final reason I think you should care. It's fun.

Yes, it can be scary. Yes, it requires thought and care when designing experiments. Yes, it will need some practice before you get comfortable and confident in designing and running experiments. But, as a break from the shopping list of features that the team has been asked to work on, I believe experiment-driven product development is, well, more exciting.

It allows you to embrace not only uncertainty but curiosity, to explore possibilities and to answer those burning questions, but in a way which is structured and has value. It's about coming up with ideas, and not getting too attached to them if they don't work out.

More importantly, it helps you, and your team, take a step back from churning out endless deliverables, and coming into work every day stressing about things which really aren't life or death decisions. It puts things into context, involves the whole team, and overall, leads to a happier, more focused, and in the long term, more successful, team.

So, there you have it, the reasons why I believe adopting this approach can help you improve your product management practice. Now, let's close out the chapter by looking at this in the wider context of an organization.

Getting the best out of XDPD in a multi-team environment

We've talked so far about experiment-driven product development in the context of a single team. Unless you're working in a startup, however, it's more than likely that the organization you're a part of has several product development teams, working alongside each other.

Perhaps you're not the leader of a single team—instead you're trying to work out, from a higher level, how this approach can work in your organization. How can you get the best out of experiment-driven product development?

Size matters

First of all, you need to consider the size of a team that adopts this approach. As with any "agile" approach, having too big a team tends to lead to inefficiencies, miscommunication, and general unwieldiness.

At the same time, too small a team leaves you no room to maneuver—the ups and downs of dealing with day-to-day reality—simply keeping the basic systems running and being one or two people down due to sickness or annual leave; never mind dealing with that moment when everything goes wrong and you need all hands on deck—all of this means you need to try and hit a sweet spot in the size of your team.

The "two-pizza" team size[5] is often a good rule—I'd advocate having at least a couple of pairs of developers, a UX pair, and at least one, if not two, data specialists (those with an aptitude for data science and data analysis).

Your mileage may vary, of course. The key thing is to try and have a team that can weather the storms while still dedicating time to the design, development, and analysis of experiments.

---

[5] As first brought to my attention via Tom Scott, in his discussion of Amazon's approach: https://derivadow.com/2007/02/20/two-pizza-teams/, though it does beg the question of what size of pizza we're talking about here….

Avoid treating an experiment team as "special"

Secondly, I'd argue against having a dedicated "experiment" team in the long run. By this, I mean designing your organizational structure around a single team who are the only ones who run experiments, on behalf of everyone else.

This is perhaps a place to start. It was the approach being used at the time I joined my first experiment-led team in 2016. The intention was to develop a deliberately siloed team, so that they could experiment with freedom, unburdened by having to maintain, or deeply integrate, with other existing systems.

However, in practice, the rest of the teams began to see our team as "the innovation team"—the ones who get to have all the fun, without any of the responsibility. Additionally, senior stakeholders sometimes viewed our team as the go-to place for any "innovative" project, which, while being a compliment, became at worst a distraction and at best a bottleneck to the company's success.

The trick is to start with a team who work in the experiment-driven way, help them to become expert practitioners, but make sure they share their knowledge with the other teams, with the aim of eventually having as many teams comfortable with this approach as possible. Make this team a "center of excellence" for experiments, and give them time and space to help coach other teams. Demonstrate by doing, but don't fall into the trap of assuming there's no responsibilities here.

Sharing what you learn

Thirdly, if you have several teams operating in this way, you need a way of sharing knowledge between them. Both in terms of the results of experiments—"we tried this, so you don't have to"—and in terms of coordinating the running of experiments, so that they don't clash and conflict.

This can take the form of regular stand-up-style updates, or perhaps a large wall display, so that everyone can see what's happening. Regular product team showcases, sharebacks, and demos can also help here—we'll discuss this in much more detail in the final chapter of the book, where I'll go through an approach to these team sessions which we developed to help share what we were working on.

Knowledge, Power, and Responsibility

Finally, you have to make a choice. Part of the beauty of adopting an experiment-driven product development approach is the ability to reduce the cost of failure—to spend less time on each "thing," and prioritize finding answers which allow you to move on. But, this can lead to an issue when it comes to the experiments that center around rapid prototyping—technical debt.

This has a couple of dimensions. If you're concentrating on developing the simplest, useful thing in order to drive meaningful results from an experiment, you may find yourselves building things fast but in an unsustainable way. The first question is—do you spend all your time just designing, building, and running experiments, forever generating useful results, but piling up the technical debt, or do you run an experiment, gather results, and then, if the experiment was successful, pay down the debt in order to make it stable and sustainable?

I'd advocate a healthy balance—run a small series of experiments, focused on gathering results, answering questions, and producing knowledge, and then take a step back. Which were the most successful variations? Which were the most valuable pieces of knowledge we gained? Allow yourself some time to take the best from what you've learned, and apply it in a "production-ready" way that benefits everyone.

Managing expectations is the other element to this—if you are a team focused solely on experiments, and with no expectation that you'll maintain successful versions of features that do well, then **make that clear from the start**, and keep reiterating it where ever possible. As I've mentioned, having such an "innovation" team has its pros and cons, but the key thing here is to make sure everyone is clear on what is expected of a team.

If you're adopting an experiment-driven product development approach within a product team that does have commitments to maintaining something, however, then that also needs to be clearly understood. It will mean that some time to pay down the technical debt will be needed, which can mean that there may be some time where experiments will have to take a back seat to simply keeping the lights on.

Equally, particularly with small teams, if the expectation is that the team is responsible for maintaining a stable product or service, in those moments where, for whatever reason, the thing you're developing may be about to fall over entirely, everyone needs to understand that at times like that, the existing thing comes first.

Remember! Make sure you have an agreement in place between the team and business stakeholders as to your priorities, and keep reinforcing this message of choices and trade-offs. Be prepared to share your reasons for prioritizing tech debt over experimentation.

Summary

In this chapter, we've explored in more detail the ins and outs of what we mean, and don't mean, by experiment-driven product development.

Experimentation is a very particular approach, and is most effectively used when it has clear direction. The exact methods by which you run your experiments—be it data analysis, multivariate testing, user research, or something entirely different, are very much secondary to the process of designing effective experiments.

This doesn't mean that you need to labor and agonize over every single aspect of the experiment design. Once you're up and running, it should become more like second nature—but simply taking some time to think about what makes sense for an experiment to be effective is important.

We've covered a number of reasons why experiment-driven product development is a worthwhile approach to consider—ranging from the creativity it affords to the importance of challenging your assumptions so that you can avoid wasting your time on things doomed to fail.

Finally, we discussed the wider organizational context that surrounds any team wanting to take a more experiment-driven approach. Sharing knowledge, and getting agreement all round, as to the scope, limits, and responsibilities of the team, is crucial—as is the need to be constantly reiterating this and making the choices involved in prioritization explicit.

With this all in mind, then, we can move on to look at how experiment-driven product development translates to some of the common activities and methods a team might use in their daily work.

CHAPTER

2

What Can You Use Experiments For?

The role of a multidisciplinary product team can vary from day to day, week to week. On a typical day, the team might be doing some market or user research, looking at site traffic trends, meeting with stakeholders to discuss future work, and perhaps even having some time together to make sure everyone's aligned on the work that's currently in progress. Oh, and let's not forget the actual development and maintenance of the product in the first place.

If we think of the product or service that you're working on as something with its own developmental lifetime, too, we can identify various types of activity that the team might engage in, in order to move things forward.

As discussed in Chapter 1, the use of A/B testing to incrementally iterate on existing features within a product is fairly well established as one of those activities by now. In this chapter, we'll look at why an experiment-driven approach works well in that case, and consider how you can apply an experiment-driven approach to some other common activities that form part of the product development process, making them more effective and useful for teams.

P. Rissen, *Experiment-Driven Product Development*,
https://doi.org/10.1007/978-1-4842-5528-5_2

If we regard experiments not just as synonymous with A/B testing, and instead take a wider definition—that which I proposed in Chapter 1—that experiments are a structured way of asking questions and gaining knowledge, we'll see that the experimental method can be applied not only to iterations on existing features but equally to

- Analyzing existing behavior within your product
- New feature development
- Resolving disagreements
- Understanding the context of your users through qualitative research

Before we discuss those, however, it's worth highlighting here how XDPD differs from, for instance, the Lean UX approach, or the similar approach made famous by the book *Sprint*.[1]

With both Lean UX and the Sprint approach, it's tempting, if not encouraged, to design solutions and use those as a way of "validating" assumptions. These solutions arise through "Design Studio" sessions which take assumptions and turn them into hypotheses—mostly, but not always, framed around a hypothetical designed *solution*.

While this certainly has its merits, the XDPD approach places emphasis on the step *before* the assumptions and hypotheses—framing every piece of work in the context of a question. In this way, new feature development, user research, data analysis, and iteration on existing features always tie back to a question that the team wants an answer to. Not necessarily a customer or business problem to be solved, but a question and an answer. With this, we ensure that we don't lose sight of the importance of discovery and learning.

Remember! In XDPD, the question is the core driver behind any experiment. We design experiments to answer questions rather than designing solutions to validate.

By the time we're done with this chapter, then, you should be confident in knowing how you might use the experiment-driven way of approaching work, to get better results. You should be inspired to start thinking of how you might apply an experimental approach to a wide range of product development activities.

In the two chapters that follow this one, we'll cover some guiding principles to bear in mind when adopting the experiment-driven approach. After that, in Part 2 of the book, we can get stuck in to the nitty-gritty of designing, running, and evaluating experiments themselves.

---

[1] Sprint, Jake Knapp, John Zeratsky, Braden Kowitz, Random House, 2016

Analyzing existing behavior

One of the key tenets of this book is to be always asking "What's the simplest, useful thing we could do to learn from?" We'll explore this in much more detail in Chapter 4, but it's worth raising now, because it's sometimes the case that if you really stopped to think about it, although new feature development or user research may be fun, it can be time-consuming and expensive. And sometimes, if you're lucky, the answers to the questions you have might already be on your doorstep.

Sometimes, if you've got a question that needs answering, all you need to do is look at how your product is already being used. You may need to make no changes at all, just to gather some data on current behaviors and draw conclusions from that. Starting with a question, working out what data you need, and deciding that with the right conditions, scale, and measures taken into account, nothing new needs to be built, can be a massive win for the team.

One approach to data analysis is to track every single piece of user activity data, all the time, just so that you have it available for analysis. This, I'd argue, is a mistake. First of all, it doesn't speak to any kind of actual usage of the data nor efficiency of having to store it all. Secondly, it builds far greater risk into your setup, as the more user data you store, the more the chances of a data breach.

While modeling and structuring the data you hold and track *are* important, storing everything just because you can is not necessarily the best road to go down.

A better approach is to apply a more surgical, experiment-driven approach. That is, add the tracking only for a very specific reason—to help you answer a question via an experiment—and for a specific amount of time. This way, you get what you need—the knowledge—while minimizing the amount of data being collected that could prove risky, or just a headache, in the future.

Remember! Don't fall into the trap of collecting user data "just because you can." Start with the question you want to answer, work out how much data you need in order to answer it, and add the tracking for as long as it takes to reach that critical mass—and no longer.

An experiment mindset to data analysis can help focus your investigations, particularly around questions that are subtly different from those that you can more easily answer with experiments that require explicit intervention, be that through a new feature or a variation on an existing feature.

For instance, let's imagine you want to understand the differences between two distinct populations or user groups, or how the existing behavior of users shifts over time. Running a "live" experiment, or doing user research, is inherently

bounded by time, and/or space, not to mention expense. The advantage of data analysis is that no explicit intervention is needed. As such, it allows you to construct a much more holistic and accurate representation of a user in situ, using your product. In due course, this can help answer questions about longer term patterns and trends in the behavior of your users.

As with any form of experimentation, by using techniques in Chapter 5 to tease out assumptions and hypotheses, you can generate a whole slew of questions that can be answered through data analysis—and, more importantly, the answers can then feed into ideas for other experiments themselves.

Another advantage of applying an experimental approach to data analysis is that you can use historical data effectively as a simulation. As long as you design the boundaries of your study using the principles of scale that we'll discuss in Chapter 7, you can simulate hypothetical changes in behavior, or in your product, using historical data, and use that as a stimulus for more questions and as a "dry run" for live experiments. These kinds of experiments can themselves help inform your assumptions and hypotheses before you do something "for real."

Fine-tuning the existing product

It would be wrong to ignore the most common "use case" for the experimental approach—iterating on existing elements of your product. This is where the standard A/B, or, if you're lucky, and you have the technical setup for it, multivariate testing comes in.

This is where most people start with the experiment-driven approach—though I'd suggest trying to run some data analysis as an experiment first, before making a chance to your product. This way, you can get used to the idea of asking and answering questions, gain knowledge, and inform future iterations of the product, without major code changes.

One of the helpful things about iterating on existing features is that you should hopefully be fairly certain of what a certain feature is designed to do—or at least, how you measure the success or otherwise of it. It's also likely that several members of your team have opinions on how features can be improved—which is fertile ground for uncovering premises and creating experiment questions.

For example—let's say you have a news web site, and one of the things you're trying to do is encourage users to read more articles in a single session. You already have a "related links" section, perhaps as a side bar. It's fairly simple to imagine that you can measure success by the ratio of unique users who click a link vs. those who see it but don't click the links.

How might you iterate on this? Well, perhaps it's the position of that "related links" section on the page. Maybe it's the title of the section itself, perhaps a different way of selecting which links to show. The point being that it's an established feature, with an established purpose, and a clear way of measuring success.

This is your classic candidate for an A/B or multivariate test. It shouldn't take long to get the different versions ready, you're deploying something to an existing part of the product, the measures are clear, and, with some techniques we'll discuss in Chapter 7, the length of the experiment can fairly easily be determined.

Breaking new ground

Performing some question-based data analysis, and then iterating on existing features or functionality, is a safe place to start with experiment-driven product development. But let's leave the shallow waters and venture out into something more risky—the world of new feature development.

There will most likely be a raft of questions, assumptions, and premises lying behind any new feature idea. Rather than simply picking an idea to work on, or worse, picking an idea and spending months building toward a vision without actually gathering any evidence that it might work, you can use experiment-driven product development to help in two aspects of new feature development.

Firstly, the question of *what* to work on. Using the experiment-driven approach, you'll learn to frame things not as feature ideas but as questions to answer. These questions can help you discover whether there is appetite for a new feature in the first place; whether the premises that would feed into your design are valid or not, through to how people seem to use a version of it; and whether they prefer it to already existing features. In Chapter 5, we'll dive deep into this process of generating and organizing questions to work with.

The second aspect where experiment-driven product development can help when it comes to new features is actually something that highlights how it can help in many, if not all, aspects of work done by a product development team. The rigor of framing your work as an experiment, and designing it carefully, can help avoid misinterpretation of the success or otherwise of what you do.

Indeed, designing your experiment appropriately really is the key to seeing how the process can be used for more than simple comparison tests. At its heart, experiment-driven product development is all about *careful design* of the way you work, to ensure that the conclusions you reach will be useful.

Make sure you're asking the right question; designing the measures, conditions, and scale of your experiment so you can get useful knowledge out of it; and *then* choosing the most appropriate method for implementation. Blindly

treating all experiments as if they have to be A/B tests, or regarding the experiment design process as only being applicable to things you build through software, is a mistake.

With a completely new feature that you're trying to assess demand for, you can use a well-designed experiment to help you learn whether or not your target audience is open to something new at all. At the very least, even with negative results, you'll still be learning what doesn't work—and, indirectly therefore, what *is* important to your users.

And if there *is* evidence to suggest you should continue developing a certain new feature, you should consider running a baseline experiment as soon as possible. These experiments, as you'd expect, seek to gather baseline data on how a feature is being used in situ. You can spend months taking into account every stakeholder, and member of your team's, advice on how best to design the feature, but the most important thing is to get something out there—the *simplest, useful thing*—a concept we'll come back to in Chapter 4.

Designing the experiment, choosing the appropriate measures and methods, will help you reach meaningful conclusions. When it comes to new features, this is especially important because of the emotional attachment that often occurs with new ideas. The results and conclusions that you draw from an experiment need to be framed in the context of the question you were trying to answer and the environment you tested within. The experiment-driven process helps you set this context, which in turn should help the chances of anyone extrapolating wild success or complete failure from any particular experiment or new feature idea.

Experiments as a way of resolving disagreement

Speaking of emotional attachment to things, it's worth noting that taking an experiment-led approach can be extremely useful when there is disagreement between team members or stakeholders over how best to achieve your goals.

People bring all kinds of preconceptions to a discussion of product development—some based on experience or expertise, some based on preexisting premises or assumptions that they have, and others purely on personal preference for aesthetics or pet project ideas.

If you're not careful, these differing ideas can lead to a seemingly never-ending cycle of discussion and disagreement, with someone, usually the product manager, having to make a decision based on whoever seems most persuasive, or worse, whoever shouts loudest. Meanwhile, resentment and unhappiness for those who didn't "win" the argument, just fester and harm the long-term health of your product team, the organization, and the product itself.

Ultimately, it doesn't really matter whose idea seems the best—you don't know, until you try. Sure, some ideas will be so out there, or so difficult to implement in the short term, that they can be disregarded, for now—though

it might be worth digging a little deeper to discover the underlying premises and assumptions that can be salvaged and considered again later. But among competing, sensible ideas, there's no better way to really decide which is better than by designing experiments around them.

Not just what, but why?

Running experiments by iterating on and developing new features allows you to determine, at scale, how your users react to your product or service in terms of what they do. Data analysis can also help you see, at scale, *what* your users are doing—how they behave under existing conditions. You can answer questions around whether people use certain functionality, which options they choose, or which paths they take—but exactly *why* they make these choices is often difficult to ascertain.

Qualitative user research, on the other hand, seeks to understand the motivations and context behind a user's actions—their worldview, and the context around them. There are several different ways to design a qualitative study, of course, but more often than not, the impetus to do some research starts with a question, a desire to understand the user more deeply.

When putting together a research brief, researchers will most likely outline the context and goals of the research, before moving on to the questions they'd like to find the answers to—these won't be necessarily the same as the questions they end up asking participants, but they will be the what the study is designed to try and discover.

Adopting an experimental mindset doesn't change much here but is a useful framework to consider and helps bring perhaps a little more rigor to the whole process, particularly in three aspects.

Firstly, when it comes to outlining the recruitment brief. The experiment process, as we'll discover in Chapters 6 and 7, helps you think about the conditions (characteristics, behaviors, demographics) and scale (number of participants, time needed for the study) that you'll need in the study to be able to draw meaningful conclusions from your research.

Secondly, you can use the framework, particularly the parts that focus on outlining existing premises or assumptions that the product team may have, to inform the questions you want the study to answer.

Although it's sometimes left unspoken, qualitative research *can* have a hypothesis included within it. It won't necessarily be a *deductive* hypothesis, but there is almost always some kind of prior belief or supposed answer that the team has, before the research has been undertaken.

Finally, and most importantly, though, the rigor of the experimental approach is extremely important when it comes to drawing conclusions from the study.

It's a common mistake to think of the design of a qualitative research study to end at "and then we ran the sessions." But extracting the nuggets of gold from the sessions is what will ultimately be of most use—and without a framework in place, research sessions can be unproductive or prone to being so vague that anyone watching can reach their own, often contradictory, conclusions of "what users meant."

Worse still, unless you're very careful about how the results of user research are communicated to stakeholders, what often happens is that people fall prey to confirmation bias—picking up on snippets of conversation from the research to justify their own conclusions.

Applying an experiment-driven approach can bring structure, not necessarily to the sessions themselves but to what it is possible to conclude and learn from them. It provides a framework to help clearly define the purpose of a study, and parts of the approach, most pertinently the techniques around premises and assumptions, can help you explore the areas of interest for your research ahead of time, which can then inform the questions you ask users. We'll cover this in more detail, in Chapter 8.

Summary

This has clearly only been a brief overview of the different activities that a product team is involved in, and how, at a high level, applying the principles and practices of experiment-driven product development can be useful for much more than just the common use case of A/B testing.

As is hopefully becoming clear, the key strength of the experiment-driven framework is around the emphasis on the question, and upon the design of the experiment itself. Refocusing your attention as a product team on the questions you want to answer, rather than hypothetical solutions to a problem, and taking the time to think about the scale needed in order to draw meaningful conclusions—all of this helps keep you attuned to learning. The *way* you learn can take many forms, and this can and should include delivery of real, working code out into the world, but the deliverables aren't the point—the question, and the answer, is.

The framework allows you to make progress in tightly defined yet meaningful and quick steps forward, ensuring that as much of your time as a product team is spent working on things that will prove beneficial to you, your business, and the users, too.

The temptation from all this may be to take this exactitude and assume that everything you do must firmly be data driven from this point forward. Alas, the real world is more complex than this. In the next chapter, we'll examine some principles to bear in mind throughout your time using the experiment-driven framework. It's something that requires not only context but humility in the team's approach.

CHAPTER

3

The Principles Behind Experiment-Driven Product Development

Experiment-driven product development is more than a particular framework or process. The method that was developed over three years grew organically, and is by no means complete or perfect. We didn't set out to define a process—we continually examined, critiqued, and refined our approach as we went, based on the principle that we wanted to keep getting better at running experiments, and thus at product development.

P. Rissen, *Experiment-Driven Product Development*,
https://doi.org/10.1007/978-1-4842-5528-5_3

Arguably, you could divide the approach into two halves—one is how to run experiments in a way that produces meaningful results (that you can learn from), and the other is how to adopt a mindset that leaves you open to a more creative, fruitful form of product development, where experiments are the manifestation of the mindset. Ultimately, it's not about picking a single idea and making it right. It's about constantly trying to learn what works and what doesn't, so that you can make progress.

In this, and the following chapter, therefore, we're going to establish some of the ground rules—the key principles behind the mindset of experiment-driven product development. As mentioned in Chapter 1 these principles share much of their DNA with those found within the agile and lean approaches. What I hope to show in this, and the next, chapter, however, is the particular lessons we learned from developing and practicing XDPD—an iteration of principles, as it were.

The key principles are

- The importance of shared understanding
- Strong opinions, loosely held
- Acknowledging, but not being held back by, uncertainty
- Being data informed, not data driven
- Simplest, Useful Thing

The last of these, Simplest, Useful Thing, will be covered in Chapter 4. The rest of these, though, is fair game for us to dig into now.

The importance of shared understanding

When the team first decided that our approach to experimentation needed improving, we hired someone with knowledge of statistics and a background in running scientific research experiments. We leaned heavily on their expertise in statistics and hypothesis development, but in doing so, we fell into the trap of dividing the team into specialisms.

The product manager and UX lead would decide the purpose and define the different conditions of an A/B experiment, the developers would build these versions, and our data scientist would define the hypotheses and the measures and do the analysis. All at slightly different times, in different rhythms.

This was a mistake that led to misunderstanding, miscommunication, and a lack of unity of purpose behind our work. Often, we'd be about to start an experiment, and we'd realize that the developers and the rest of the team weren't on the same page about what exactly needed to be built.

Sometimes, everyone aside from the product manager would be confused about why exactly we were running the experiment in the first place, other than it being "a cool idea we want to try." And no one, except the data scientist, really seemed to understand the link between statistical significance and the duration of an experiment (something we'll cover in much more detail in Chapter 7).

Having team members with knowledge and experience of testing and business analysis helps, of course, but ultimately the problem lays in the disconnect between the disciplines and indeed between the team and the stakeholders, too.

What we'd failed to realize was that the success of an experiment—the chances of securing a useful answer—depended on **a shared understanding among the whole team**. It needed the involvement of as many members of the team, across different disciplines, as possible.

I've seen a similar issue emerge in larger, more traditionally "lean" teams. It's very easy to treat, for instance, the user experience specialists on a team as separate and different from the developers. The former are regarded as being the ones who want to do research, in order to understand and design for the user, while the latter are charged with delivery and improving technical performance.

What this undersells is the true nature of technical development, which is as much about trial and error, about research, as anything else. Rather than inviting, or worse, trying to cajole developers to observe user research sessions, assemble multidisciplinary teams around trying to answer a question, so that everyone has a shared understanding of what a particular piece of research will involve.

What we eventually put in place was akin to a "story kick off"—an "experiment kick off," as it were.

The Experiment Kick Off

Before starting work on an experiment, we would sit down, as a team, and collaborate, through discussion, on writing down the most important details of the experiment:

- What are we trying to learn?
- Why is it important for us to find this out?
- What are we going to do?
- What change or difference are we expecting to see?
- How will we measure success?

- What will be the conditions for participants in the experiment?
- How long will we run the experiment for?
- What subset of possible participants will we include?

These points would be captured on an index card—what we called our *experiment card*. This could then be used as a reference point for anyone in the team during any part of the development, running or analysis of an experiment.

We'll look at the experiment card in much more detail in Part 2 of this book, but the key point to understand for now, though, is that sitting down and having the discussion as team, writing the card together, were of huge benefit.

Why? Because it's crucial for **everyone** on the team, and ideally stakeholders as well, to have a clear understanding of the rationale behind and the details of an experiment. Similarly, it's really important to make clear, in a succinct way, exactly what the experiment will need in order to be successful.

With larger teams, it may not always be possible to have everyone in the discussion—you should at least ensure that your discipline leads, if you have them, are involved. It also helps if you've used the prioritization process described in Chapter 5 to determine which experiment to run—as it's likely that more team members will want to contribute when an experiment involves a question they really want to know the answer to.

Remember! Involve the whole team, or at least your discipline leads, in writing the experiment card. The earlier you can establish a shared understanding of an experiment, the more likely it will lead to a meaningful, useful answer.

With a clear purpose in mind, members of the team involved in preparing and running the experiment must agree and specify details around exactly how evidence will be gathered. Stating what, if anything, needs to be built, what should (and shouldn't) happen under different circumstances, in a place that everyone can refer to, is crucial.

Finally, being clear about what can, and can't, be learned from the experiment—how long it needs to be run, what that means, and what the results tell us.

Having this "kick off," allowing time to ask questions, to nail down exactly what we were trying to learn, what our hypothesis was, and what the conditions would be, was really helpful, making sure that everyone not only knew, and agreed upon, what we were, and weren't going to do, but why.

Everyone came to this with different perspectives—different things they specifically cared about, both personally and from the point of view of their discipline. Sharing these openly, and agreeing the boundaries of the experiment up front, made us far more productive and helped us learn.

It also helped us when working with stakeholders, too. This was another step in the evolution of the process, but we found it just as important to communicate clearly the what, and why, of our experiments, during our regular showcases, linking back the reasons for running a particular experiment, to our wider goals.

Showing, and discussing, the experiment cards themselves with stakeholders, meant they were clear and on the same page as the team, too. They didn't necessarily need to know the exact details of how, but they wanted to understand why and, especially, what we'd learned.

This, in turn, meant that we morphed from a team responsible for "getting an idea right" to a team that could provide intelligent, useful feedback to the business, about product ideas, about how users behave—which itself meant that ideas that didn't seem to work could be abandoned with little cost, and promising ideas developed. It was only by treating the entire process as a team sport, however, that got us to that position.

Strong opinions, loosely held

Ideas are cheap. Good ideas, *those* are the hard ones to find. Ideas are also attractive, and we can get possessive about them. The idea you come up with, you tend to stick to, even if evidence tells you that it might not be as correct as you think. This is a common cognitive bias within humans, and in order for a team to be effective at experiment-driven product development, it needs to be acknowledged and combatted explicitly.

This really does come down to mindset, both in terms of the team members and from a business as a whole. It's a radically different way of looking at how any project within an organization progresses. And it isn't easy. As humans, it's hard to deny our egos—to have an idea, and to be so sure of its value, that we'll pursue it at all costs.

It's closely related to a common issue within businesses, that of "solutionizing": as professionals, paid to do a job, we want to find problems and solve them. As we become more experienced, we rely on that experience to come up with solutions. And then, often, we take the leap to fixate upon a particular solution, attaching our success and value to that idea.

Undeniably, there's a good intention at the root of this—we want to be helpful; we want to feel as though we've contributed to solving a problem. But this attachment to a particular idea often leads to pain. If it turns out that the

particular idea, or solution, doesn't seem to, in reality, do the job, it's easy to blame things. It's easy to regard it as a judgment on your ability, a stain on your reputation. This is wrong.

A core principle of experiment-driven product development, therefore, is to try, as best as you can, to remove the emotion and attachment to specific solutions. To have strong opinions, but hold them loosely. If they seem to be wrong, or misguided—there should not be any shame in that. There is no problem, when confronted with evidence, to say "I was wrong about this." Again, the sooner you can tell whether you're right or wrong about a particular idea, the sooner you can stop wasting time fixating on ideas that don't work.

Of course, it's rare (though not impossible) to run a single experiment and be able to tell, definitively, that something is a "good" or a "bad" idea—the important thing is not to be too attached to the idea itself. Indeed, trying as much as possible to stop testing ideas, and instead concentrating on answering questions, and testing the premises *behind* the ideas, can prove much more useful. Setting clear boundaries, when defining an experiment, about what aspect of the idea you're testing, and what you're able, and not able to, learn from it, also helps remove the emotional attachment.

■ **Remember!** Avoid fixating on solutions. Concentrate instead on the premises that underlie those solution ideas. Always be asking questions and seeking answers.

Ultimately, it's all about placing the importance on the question to be answered. Seeking an answer, making it explicit if you have a prior belief, gathering evidence in a way that will gain you useful knowledge, and seeking to test whether the evidence supports or contradicts your belief. Regardless of the result, the outcome of an experiment shouldn't be regarded as a personal vindication or judgment on your worth.

This goes both ways, too. A common trap is to fall into an "us vs. them" situation between a team and their senior stakeholders—the idea that good ideas can only come from the team, and everyone else is just ill-informed. But the evidence doesn't always bear this out. Sometimes the team is right, sometimes they're wrong.

For example, one day, a message came down to the team from a (very) senior stakeholder who had seen our recommendations-by-email service, asking us to offer this on a daily, rather than just weekly, basis. Initially, we scoffed—we knew that the recommendations weren't likely to change so quickly, so why would anyone want a daily email service?

Nevertheless, we came up with a quick way of testing whether people would prefer a more frequent email service—a demand test—rather than building the entire service (or indeed disregarding the idea completely). And we were

wrong, there *was* meaningful evidence to suggest that this might be something people wanted. We had learned something—not that every idea that comes from above is necessarily correct, but that testing basic ideas, cheaply, can help you spot things you've missed.

Ultimately, it didn't matter who was right or wrong. What mattered was trying the thing—and being clear about what, exactly, the results could reasonably tell you.

Acknowledging, and embracing, uncertainty

What you can learn from a single experiment is, by nature, limited. For an experiment to be useful, we have to be able to run it within a reasonable amount of time. For an experiment to be meaningful, we need to ensure that enough data is collected, from which we can draw conclusions. But both of these criteria aren't always possible, in reality, and we can't avoid the fact that in the real world, there are millions of factors beyond our control. We do not have the luxury of completely clean, test environments where everything is a blank slate, and no one stumbles upon an experiment in the wild without preconceptions.

Therefore, what we can learn from a single experiment has its bounds—if we were to run the same experiment again, it's possible we could have a different result. If an experiment shows that an idea seems to work, we must keep perspective in mind, and not congratulate ourselves for having solved something definitively. If it shows that the idea isn't so good, we can take solace in the fact that this negative result holds true under very particular circumstances.

Potentially, this has the potential to render experiments meaningless. The very opposite of something that we can learn from. If every experiment you run ends up concluding that what can be learned is only applicable within a very narrow set of circumstances, is it really that useful?

Similarly, if any negative result can be explained away by it not being the right circumstances for the idea to take flight, then this can reinforce people's attachment to, and unwillingness to give up on, certain ideas.

No experiment will ever be perfect. We can never be 100% certain of something. And yet, that doesn't mean that all of this is fruitless. What it means is that we need to recognize the importance of three things:

Firstly, it is vitally important to **design your experiments**. To make sure that you can, within reason, gather enough data, at a meaningful scale, so that you can make reasonable conclusions, with enough control, from which to learn.

Secondly, if it is *not* feasible to reach a meaningful conclusion with enough precision, explicitly communicating to others that what you learn is true under the conditions you were able to run the test, and that the conclusions you draw have limits on the precision of certainty, is really valuable. By doing this, not only can you set expectations, but you can also inspire new questions, new experiment ideas:

- We know that X holds true under Y's circumstances, but what about in the case of Y?
- Is there some other way we could test just Y's circumstances?
- Are there ways in which we can refine or hone our experiment definition, to learn something even more useful?

Thirdly, if something produces a negative result, **it hasn't failed**. It doesn't mean the thing you were experimenting around was a bad idea, and will always be so. But equally, it doesn't mean that there is no evidence to suggest it *might* be a bad idea, under certain circumstances.

Again, given that the evidence doesn't seem to support a hypothesis, ask yourself how can you refine or hone it, to see under different circumstances, whether it holds true. **There are no failed experiments—there are always lessons to be learned and inspiration to be gained.**

Sometimes this is just a case of waiting a while, and running it with different people, at a different time of year. But at the same time, you don't want to be endlessly pursuing an idea until you find some evidence to support the hypothesis. Seeking evidence that confirms your bias, without acknowledging the limits of certainty around that evidence, is just as harmful. It has no basis in scientific study, and more importantly, from a business perspective, you end up throwing good money after bad, in order to support someone's ego.

Remember! The evidence you gather drives the lessons you learn, not the superiority of ideas.

Be data informed, not data driven

It's sometimes said that "there are no good or bad ideas, only data". From someone advocating an experiment-driven approach, this would seem sensible. But this is wrong.

Gathering evidence, and data, is important, but we do not live in a vacuum. Almost everything we do has financial, societal, political, ethical, and moral implications. And sometimes, even though "the data" may point in one direction, you need to take a step back and ask yourself—is this the right thing to do?

There's a story told by historians, that in the 1950s, there was an influx of ideas from statisticians and economists, into historical practice. Members of historical research institutes were delighted. Using these new tools and methods, they could take a step above dealing with examining individual stories and start to examine economic patterns in human history at scale. Number crunching machines could help them assess large-scale movements for the first time. A new discipline within historical inquiry was born—cliometrics. But, in the rush to do so, to quote Dr. Ian Malcolm from the original Jurassic Park, they "were so preoccupied with whether they could that they didn't stop to think if they should."

Such was the case with analyses of the Slave Trade, not once, but twice. In 1958, a pair of economic historians published an article entitled "The Economics of Slavery in the Antebellum South"—arguing that, based purely on statistics, slavery had been a profitable and efficient system.[1] This was followed by "Time on the Cross," a book with a similar argument, by different authors, which generated huge controversy.[2]

Both sets of authors probably didn't intend to be seen as passing moral judgment on the system, merely stating from a dispassionate viewpoint the economic facts of the matter. The fact that they performed the analysis wasn't the issue. It is inarguable, though, that such statements had to be made with caution and responsibility.

Even if it *were* true, from a purely neutral statistical and economic point of view, drawing conclusions solely from the data in such an emotive case, one laden with societal implications, would have been a terrible idea. Data driven does not always equal the right thing.

The conclusions you draw from something, be it data analysis, user research, or something like an A/B test, can only tell you so much. Whether something is "good" or "bad," cannot be derived from the data—it is your responsibility to make that judgment call.

Many teams, many companies, pride themselves on being "data driven." If we take them at their word, and many teams who try to implement some form of experiment-driven product development do, then being "driven" by the data results in implementing "successful" ideas, no matter whether they are the right thing for a team, a company, or society.

---

[1] *The Economics of Slavery*, Alfred H. Conrad and John R. Meyer, Journal of Political Economy Vol. 66, No. 2 (Apr., 1958), pp. 95–130 www.jstor.org/stable/1827270

[2] *Time on the Cross: The Economics of American Negro Slavery*, Fogel, Robert William & Engerman, Stanley L., W.W. Norton and Company. ISBN 978-0-393-31218-8. Reissue edition 1995; first published in 1974

Just chasing the numbers can lead you down very unsavory paths. Experimenting, gathering evidence, and collecting data—these are vitally important and useful, but ultimately, the responsibility for what you do with that, for what happens next, and for how you proceed, should rest with yourself and the team.

Plenty of times, you won't need to worry about this too much—what the data tells you is almost certainly fine—and you can course correct later, if needs be. But it *is* a key principle that you should always remember.

■ **Remember!** You can be *informed* by data, but you should be very wary of ever being totally, and solely, *driven* by data.

Summary

In this chapter, we've covered most of the key principles of experiment-driven product development. The philosophy, if you will.

If you take nothing else away from this book, if you try the specific approach of experiment cards and the suggestions for sharing back with stakeholders, and they don't work, that's fine. Create your own version of the process. Refine it. Never stop trying to improve it. But above all else, remember these points:

- Being experiment driven means not just delegating to statistical experts. Involve the whole team, and seek clarity of agreement.
- Be humble with regard to ideas for features. Instead, frame these ideas as experiments, questions, and things you are curious about and want to learn more.
- Regardless of the result of an experiment, don't get carried away—separate ego from evidence.
- Just because a positive or negative result is obtained, doesn't mean that it is the right or wrong thing to do.
- The responsibility for whether to proceed in a particular direction, based on the results of an experiment, lies with you, with the team.
- Just because you can doesn't mean you should.
- Take care to design your experiments, set boundaries on their validity, and communicate that clearly, so that not only can you learn something useful but that everyone is aware of the limits of your knowledge.

- Use those limits to explore, and learn, further.
- Above all, be data informed, but not data driven.

Now, the preceding points are useful on a theoretical level, but let's start getting practical. Given these principles, how do you actually approach the work, regardless of whether it's building, researching, or analyzing something? How do you decide what a piece of work should be?

The answer? Our final key principle—Simplest, Useful Thing.

CHAPTER

4

Simplest Useful Thing

In the previous chapter, we covered a set of principles that should be in the back of your mind during the development of a digital product. The importance of shared understanding among team members; having opinions, but being prepared to adapt them in the face of evidence; embracing uncertainty; and being data informed, not data driven.

But the greatest of these principles is the one we didn't yet cover, because it deserves its own chapter. The principle of the Simplest Useful Thing. In this chapter, we'll

- Compare "Simplest Useful Thing" to its famous cousin, "Minimum Viable Product" (MVP)
- Identify some of the different ways people understand the MVP concept, allowing you to recognize and prevent time-wasting arguments over a definition
- Deconstruct the idea of Simplest Useful Thing in order to examine its different aspects

P. Rissen, *Experiment-Driven Product Development*,
https://doi.org/10.1007/978-1-4842-5528-5_4

What do we mean by MVP?

Many people reading this chapter will probably be familiar with the concept of "Minimum Viable Product" (MVP). However, if you asked ten people working in a business that developed digital products and services to define what an MVP was, you'd probably end up with at least five, if not more, different definitions or opinions over what it actually means.

Different disciplines tend to have different understandings of the concept, and frankly, if you're spending more time at the beginning (or indeed at any stage) of product development furiously debating among team members as to whether what has been, or worse, what might be, built qualifies as an MVP (or even whether an MVP is something you should be aiming to build), you're doing it wrong.

Deconstructing the MVP

The sensible thing to do, therefore, would be to go back to the source material, the original definition of MVP, and examine that, in order to reach a better understanding of what it means. But before we do that, let's look a little closer at four ways in which people interpret the concept. In doing so, you'll be better equipped to spot where theoretical debates are in danger of causing time-wasting black holes of disagreement among your team and stakeholders.

Minimum and Viable

In this interpretation, the first two words of the phrase are emphasized. That is, that whatever you build merely needs to be functional, to be something that you can release to users.

On the plus side, it's good to know that this school of thought would not want to release something that is broken—something that isn't viable. At least we all agree that giving the user something that doesn't work is a bad thing.

The down side, however, is that what constitutes "minimum viable" isn't clear. What might appear to be functional and obviously working to the team may not be clear, understandable, or usable by the public. If you set your standards that low, where does it end?

Often, I've seen this surface in internal debates between user experience specialists, product managers, and developers. The concern is that too much emphasis on delivery ignores the need to research and design solutions which will be acceptable to and used by people in the world. This is valid if the emphasis is on delivery for delivery's sake—but it's a key reason why concentrating on the deliverable—the prototype, the code, or the sketch—is misguided.

Concentrate on what you want to learn—get something, *anything* out there in front of real users, in some form, *so that you can gain knowledge*. Delivery should be the *means* by which you research and learn, not the end in itself.

Focusing purely on delivery and functionality also ignores the importance of form and user experience, particularly if no one is thinking about the bigger picture—you might end up releasing hundreds of minimum viable features or products which by themselves do the job but are painful to use and, taken together, add up to a product no one would choose to use.

A "Viable Product"

In this second interpretation, the latter two words are emphasized, in order to concentrate on what constitutes something that will "work" in the sense that people will choose to use and, if it's a commercial product, buy.

The upside here is clear—by taking the time to think up front about the market that you're entering, competitor products, user needs, gaps in the market, and so on, you can have a pretty good idea beforehand about what it is that your product needs to do, as a baseline, to at least sell a respectable number.

It's clearly important to do your homework before, and during, product development. At the end of the day, however, you'll never know for sure whether something will succeed with users until it's actually in their hands. In real-life circumstances, with real data coursing through its veins, that's when the "market fit" comes into its own.

The danger here is that it's very easy to get stuck in analysis paralysis, endlessly analyzing market trends, debating, and worrying about whether the thing you're building will be the "killer app" for your market. Nobody knows. There are no guaranteed successes. Treat your market research as an experiment—focus on the question, work out the simplest useful thing you could do to answer the question, and proceed from there.

"Viable" means "Acceptable"—but to whom?

The third interpretation seeks to clarify what is meant by "viable"—and particularly, frames that in terms of what should be acceptable, both to the team and to users. It becomes a question of professional pride, as well as user satisfaction—the former giving this a particularly volatile sting when it comes to team discussions, as it can appear to be attacking the reputation of team members themselves.

This all stems, of course, from a desire to look out for the user—to deliver something that isn't just functional, but is well designed and fits in to their lives. It contends that there should be certain minimum standards of design which must be met before something can be released.

A noble goal, for sure, though agreeing and clearly defining these standards, let alone the rules around exceptional and edge cases, can be a huge task in and of itself—should it include accessibility standards? Performance considerations? Almost certainly all of this needs to be considered, but this almost needs to be done as a project alongside product development. Agree what you mean by "viable"—your minimum acceptable principles—as soon as you can.

Product—The Maker's Privilege

The final interpretation rests upon the word "product" within MVP. Again, the use of this word stems from good intentions—make something, and get it out there into the real world, somehow, so you can gather real feedback. However, the assumption implicit in this is that you must *make* something.

The insistence that you can *only* ever learn from designing, and making, a solution, regardless of fidelity, I'd argue, sets us back into the mindset of solutions and deliverables, over and above asking and answering questions. Sure, a good MVP should help you learn, but I've seen too many teams leap to designing a solution, such that the knowledge that arises after the fact is rendered either coincidental or open to so much interpretation that it doesn't help make a decision on future direction clear.

Where does this leave us?

It's clear that "Minimum Viable Product" is a loaded and hotly contested term. The upshot of all this is that while each interpretation (and I'm sure there's probably plenty others not outlined earlier) has valid concerns and interests, they're also each problematic and, worse still, can distract from figuring out what the most useful thing would be that could help you answer a specific question.

The OG MVP

The term "Minimum Viable Product" appears to have been coined by Frank Robinson, and later popularized along with the idea of the *Lean Startup* by Eric Ries.[1]

[1] *The Lean Startup*, Eric Ries, Crown Publishing Group, 2011

Robinson defines it in a very economical sense—an MVP is

> *...that unique product that maximizes return on risk for both the vendor and the customer.*
>
> —Frank Robinson, A Proven Methodology to Maximize Return on Risk[2]

Ries, in contrast, frames it in terms of a team, learning

> *...the minimum viable product is that version of a new product which allows a team to collect the maximum amount of validated learning about customers with the least effort.*
>
> —Eric Ries, Minimum Viable Product: a guide[3]

Gothelf and Seiden consider it from a designer's perspective:

> *…We defined the Minimum Viable Product as the smallest thing you can make to learn whether your hypothesis is valid.*
>
> —Jeff Gothelf and Josh Seiden, *Lean UX, 2nd Edition*[4]

Note that the first two definitions emphasize *minimum risk* in return for *maximum return*. The definition of "return" is still subjective—you could interpret Robinson's definition in terms of purely economic return, whereas Ries brings the knowledge being gained to the fore. The third definition reinforces my point that it's the word "product" which is doing the most damage here, because it concentrates the mind on a "finished" thing, rather than what's actually useful, regardless of form, to the team, the stakeholders, or ultimately, the user.

The problem here is that, particularly in an "agile" development world and especially in the realm of experiment-driven product development, it's too easy to either get stuck in endless debate over what constitutes the minimum viable product or to use the idea of an MVP as a safety net to continue thinking only of delivery, not learning.

Admittedly, this is a worst-case scenario—and there are some minimum standards such as accessibility and performance that I'd want to defend wholeheartedly—but if you're setting a team up to approach product development in an experiment-driven way, it's almost as if you need to set the whole idea of MVP aside.

---

[2] www.syncdev.com/minimum-viable-product/
[3] www.startuplessonslearned.com/2009/08/minimum-viable-product-guide.html
[4] *Lean UX* (2nd Edition), Jeff Gothelf & Josh Seiden, O'Reilly Media, 2016

This doesn't give you an excuse to chuck inaccessible, badly designed, ill-conceived ideas out to the market that have no hope of surviving. Deciding on what you, as a team, hold as your minimum acceptable level of work, as well as making sure you do your homework and understand your users and the market, is a worthwhile and important thing to do.

It's not that MVP is a terrible idea, but if you approach product development in a humble way, acknowledging that, even if you have your own acceptable standards of work, you don't know what will succeed, or indeed what you will learn as you proceed, it's simply not a helpful way of framing things.

Which leads us on to….

Defining Simplest Useful Thing

Just like MVP, Simplest Useful Thing boils down to three elements. Defining what we mean by each element is crucial. Similarly, when undertaking a piece of work, it's important to reach agreement among the team and communicate clearly to your stakeholders, how you're defining each element for a particular piece of work. So, let's take each word in turn.

Simplest, rather than minimum

Wherever possible, you want to ensure that what you do, in terms of the amount of effort and complexity, both for the team and for users, is simple. During the development process, you should always be asking—is this the simplest thing we could do and learn from? If you *are* building something, then ask—is this the simplest thing for a user to understand?

Simple doesn't always mean the first solution that comes to mind—that can imply a lack of care in the work produced. What's important is that the way in which you try to test a hypothesis or answer a question—that should be done in a way that gives you a valid answer from which you can learn but doesn't overcomplicate things.

The minute that you find it's taking weeks just to get an experiment up and running—be that organizing user research or building a feature to put in front of users—you really have to question, *is this really the simplest thing we could do, that we can learn from?*

Sometimes, rather than building a prototype of a feature for an A/B test, it would be quicker, cheaper, and simpler, to go and do some user research, or, if you have a dataset already, analyze what you have—any activity that will give you a useful answer, one that you can learn from, is fine. Remember, experiments are just a way of asking and answering questions—they don't always

have to involve A/B or multivariate tests. Consider what you already have to hand—be that access to APIs, datasets, or tooling—what can you reuse, in order to help you answer the question?

If it does turn out that building a version of a feature is the simplest way that you could get a useful answer, then it's still worth asking, *what is the simplest version of this feature we could build, in order to get a useful answer?*

Let's take an example from a project I consulted on. A newsletter product might wonder whether their audience would like to receive the newsletter at different times of the day, to suit their tastes. Perhaps they have users around the world, and the current setup sends the emails out at 9am UK time, regardless of when that might be for users around the world.

One way of testing this might be to build a feature that sent out the newsletters at different times of day and see which got the most engagement, using measures like open rate and/or click-through to content from the email itself. But building a system that takes into account time zones, or randomly sends emails out, isn't exactly a simple thing to do.

Instead, think about the simplest way you could answer the question. How might we ascertain whether there is even any demand for such a feature? Before you spend time and effort building the functionality, why not put a link in the emails as they stand, and see how many people click a link that says something like "want the newsletter delivered at a different time?" You wouldn't even need a huge sample size to be able to get an initial idea of whether it was a good idea or not.

There's parallels here to the "minimum" in MVP, but the key difference is that "minimum" suggests an agreed threshold of acceptability, which, as we've discussed, is often more complex to define than it first appears, and can get you mired in weeks of endless debate and development, as you struggle to get sign off from all parties. Your standards for acceptability should be agreed at a team level rather than at an experiment level.

In contrast, "simple" keeps you focused on the method rather than the effort or the final deliverable. It's still important to ensure what you do will give you a valid answer from which to learn. It's also important to ensure that whatever you do, the potential for causing harm to your current users, or product more generally, is minimized as far as possible. Keeping things simple keeps the risk small—but still allows you to take risks.

Remember! By simple, we mean the simplest way of answering the question—which won't necessarily be the first thing that comes to mind.

Useful, rather than viable

As we've seen, "viable" is another of those contentious words. Useful, however, is anything but. You don't know, up front, whether an idea is going to be viable. What you can ensure, however, is whether the thing you do is useful. It can be *useful for you as a team*—useful to learn from, useful to clarify, useful to gain knowledge—or it can be *useful for the user*. But it has to be at least one of these.

Knowing that something is going to be useful for a user, ahead of time, is just as tricky as knowing whether something is going to be viable. Which is why it's so important to reframe the work of product development in terms of questions and premises—you may believe something will or won't be useful for a user—*so go and find out*. Even if it doesn't turn out to be useful for a user, running the experiment, answering the question, will have still been useful for the team as a whole.

What's important here is to explicitly connect the work you to do *how it is useful*—be that for the team or for the user. Stating clearly what a piece of work will allow you to learn, and what you did in fact learn, after the experiment, will help both those inside and outside the team understand the utility of the work you're doing and will in turn both help you avoid wasting time on pieces of work that are black holes and inspire new questions and new experiments to run.

From a user-focused point of view, it's worth thinking too, about simplicity and usefulness. What is the simplest, most useful thing that we could provide to users—something that would solve a problem, do a job for them with the minimum of fuss? When we talk about innovation, as Jared Spool has noted, really we're talking about providing users with something that would really, really help them—it can be a small tool, which dramatically helps them.[5] If it fills a need, solves a problem, saves time—if it genuinely helps users, it can be as simple as you like.

Finally, the concept of "utility" here also includes how useful the answer to an experiment can be. This really boils down to how meaningful the answer is—what you can extrapolate from it—and this, in turn, comes down to the questions of scale which we'll be discussing in Chapter 7.

Given the size of the population that you have access to, be that in terms of advertising, data analysis, user research, or actual users of your product or service, how many people would need to be included in the experiment for you to get a useful answer—one that can be relied upon to help you in your next decision. And if you need more people than would be possible in a reasonable amount of time, then working out the certainty of the answer you can get from the experiment will help you determine the usefulness of running the experiment in the first place.

[5] https://articles.uie.com/innovative_experience/

Remember! An experiment can be useful to the team, useful to the user, or both. The former means that the answer has to be legitimate enough to provide a way forward. The latter means that whatever results from the experiment should have utility for users.

Thing, not product

An MVP focuses a product development team on getting something out the door. This isn't a bad thing, by any stretch of the imagination, but unless you're making something physical, even the word "product" has a very wooly definition. Does this mean that you can't release a single feature, it has to be a whole "product" release? Where does user research and data analysis fit into here, never mind all the other activities that a product development team undertakes? Why does everything have to be focused on *building* something?

Now, don't get me wrong, "shipping" something (and I don't mean fan fiction!) is a worthy cause—getting something into the hands of your users, so they can benefit from your hard work, and you can learn from how people *actually* use the thing you've made, is definitely a good thing. But the obsession with "product" means that the hard work of the other disciplines, aside from software development, can be ignored, shunted into some form of "sprint zero," or worse, treated in a waterfall manner—do it once, then forget about it.

At the end of the day, as a product manager, I don't really care whether we've shipped something or not—I care that we've produced something, that we've completed something, or rather, that we've *learned* something. There's going to be days where everyone is in the middle of something, and there isn't a clear output yet, and that's fine, but that work should always be guided by an end goal—a state by which you'd be able to say "as a result of doing X, we learned Y."[6]

Equally, you should acknowledge that the developers on your team, whether or not they are building prototypes or parts of the "real" product—are just as much engaged in the practice of experimental research as anyone else. Not everyone has to express themselves in sticky notes and felt tips.

We'll cover more of how you can clearly communicate your end goals, and achievements, to stakeholders in the final section of this book, but for now, the key takeaway here is that despite the name, a product development team doesn't *only* develop products, and it's time to make that official, and accepted.

[6] There are some activities, such as ongoing maintenance, or just keeping something running, which won't necessarily have such clear cut-end goals. Those should happen anyway, though it may be useful to consider which of those might be made more effective if they did have clear end goals.

■ **Remember!** Don't get hung up on the need to deliver a solution for every experiment. Instead, do the simplest thing that can get a meaningful answer, ideally by interacting with real users.

Summary

In this chapter, we've examined the concept of "Simplest Useful Thing" and compared it to the more often discussed concept of "Minimum Viable Product."

Minimum viable product may be something that you're building toward. If you're going to use that terminology, make sure that everyone is agreed on what the criteria for "minimum" and "viable," and bear in mind that once released, what you've built may not turn out to be viable after all.

In the process of building *toward* the MVP, however, I'd propose it's more useful to use Simplest Useful Thing as your guiding light on a day-to-day basis. Always be asking—what is the simplest, useful thing that we could do next, which would allow us to learn; take us closer to where we think we want to go; and ideally, be simple, and useful for our users, no matter what form that takes.

Most importantly, remember

- Concentrate on gaining knowledge—testing your premises, answering questions, rather than delivering product and feature ideas for the sake of it.
- Always be asking—is this the simplest thing we could do that would answer our question?
- Is this the simplest, most useful thing we could provide for our users?
- Do we have anything to hand already, which will allow us to answer this question in a simple, useful way?
- How useful will the answer be, given the constraints of the population available to us?

And that's it! We've now covered all the principles and core tenets of philosophy which form the bedrock of the experiment-driven product development approach. Armed with all of this, we're ready to get started.

In Part 2 of the book, we'll cover the process—how to generate ideas for experiments, how to define them, and what to bear in mind when setting them up to run. And in the final part, we'll go over what to do in the aftermath of experiments and how to communicate progress.

PART

2

Experiment-Driven Development in Practice

CHAPTER

5

Getting Started

In Part 1 of this book, we explored the ethos behind experiment-driven product development (XDPD). Now, armed with this knowledge, and with the principles such as "simplest, useful thing" echoing in your mind, we can dive into the practical side of things. Before we get started, let's take another look at the basic XDPD process I outlined in Chapter 1 (Figure 5-1).

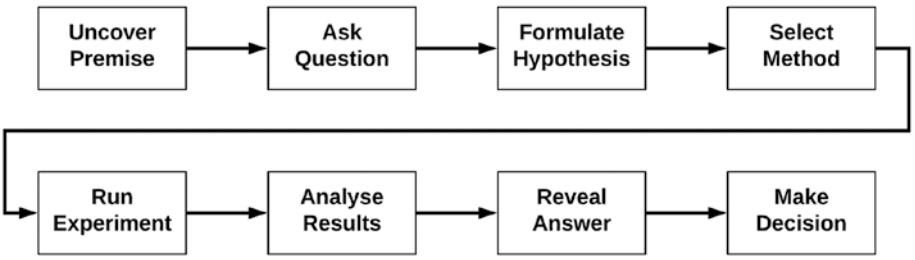

Figure 5-1. The basic process of experiment-driven product development

This high-level view covers the overall flow of the process, but in Parts 2 and 3 of this book, we can go into more detail. XDPD can be broken down into essentially four phases—planning, designing, running, and analyzing.

In this chapter, we'll discuss how you get off the ground with XDPD—what I call the Planning Phase—and return to the concept of the "Experiment Card" that I introduced in Chapter 3, as we prepare to design an experiment for real.

P. Rissen, *Experiment-Driven Product Development*,
https://doi.org/10.1007/978-1-4842-5528-5_5

The Planning Phase

The Planning Phase is where it all starts. In this phase, we take various forms of raw material and seek to uncover the premises that underpin them. From that, we prepare "experiment-ready" questions, and establish our motivation for seeking an answer. Finally, we put together a prioritized backlog of questions to work through. See Figure 5-2.

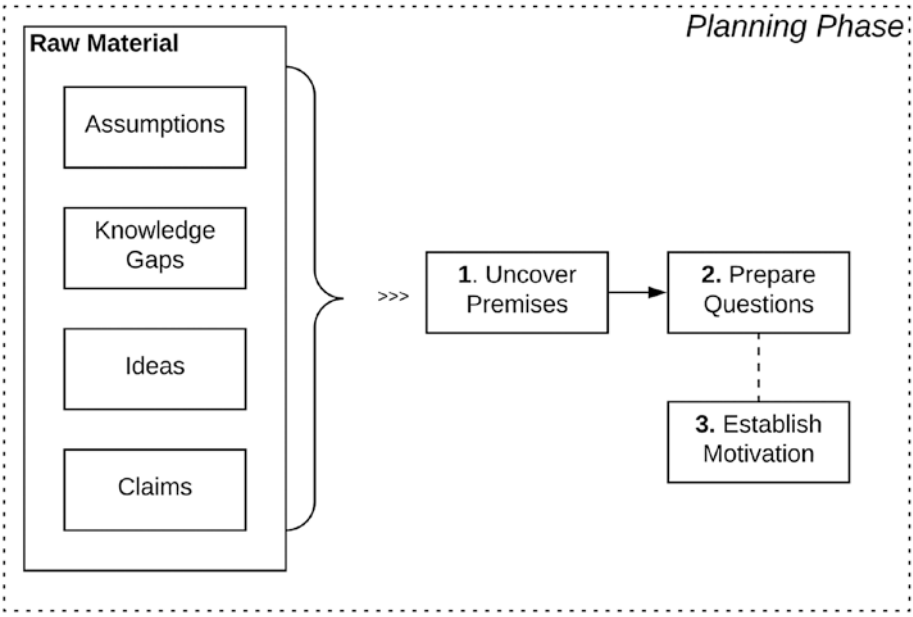

Figure 5-2. The Planning Phase of XDPD, where you take raw material, uncover premises, prepare "experiment-ready" questions, and establish our motivations for answering them

The importance of the question

It's all very well, wanting to be experiment driven, but blindly applying the elements we'll discuss in later chapters, without taking a moment to step back and consider *which* experiments you want to run, can lead to directionless, wasteful teams and work. Similarly, trying to superimpose the experiment process upon a more traditional backlog of feature development, while it can help somewhat, only gets part of the benefit out of the approach. As we established right at the start of this book, experiments are much more than just A/B tests, or a way to test potential design solutions. *They are a structured way of asking questions*. Questions themselves are things that we want to know the answer to.

Depending on what phase of product development you're in, you'll have different kinds of questions. When in the initial "discovery" phase, you'll want to know more about the situation that your potential users find themselves in. As the product develops, you'll continue wanting to explore this, but you'll also likely want to find ways of making small improvements, in the direction of a larger goal. And, of course, you should always be trying to understand how your users are currently behaving when they do make use of your product, so you can iron out bugs and uncover new ways of helping them.

Rather than thinking of a backlog as a prioritized list of features or solutions to test, you'll want to start thinking of it as a set of questions for research.[1] Framing everything as a set of questions can help you avoid (though not eliminate) the issue of being too focused on developing certain, already agreed solutions—and instead allowing the freedom to develop the *better* solution.

Remember! In XDPD, the question has primary importance. This is in contrast to other Lean approaches, which emphasize hypotheses or proposed design solutions as the starting point for experiments.

It can be relatively easy to sit down and come up with a list of questions you want to know the answer to, regarding your product or service. Simply starting from a blank sheet of paper, however, can be quite unhelpful. For one, you're likely to end up with a set of questions that traverse the spectrum from extremely specific to so broad they may never be answered in millennia. Equally, sometimes it can be maddeningly difficult to think of relevant questions.

You need a way to focus the team's thinking. Picking a particularly interesting or important customer or business problem to concentrate on is one way of approaching this.[2] Another way, if you are using the "Objectives and Key Results" (OKR) framework, is to focus on a particular objective and at least one associated key result that you're trying to work toward.[3] However you choose to focus the team's attention, you want to find the things that the team are most curious or passionate about.

---

[1] Of course, in reality it may not be easy to simply switch from an existing traditional backlog, to one filled with questions—but think about shifting, gradually, from one to the other if possible.

[2] This is the approach favored by the authors of *Lean UX*, for example.

[3] *The Art of the OKR*, Christina Wodtke, 2014, http://eleganthack.com/the-art-of-the-okr/

Assembling the raw material

Inspiration for questions can come in all shapes and sizes. In order to generate the questions, we need to assemble a collection of raw material from which all kinds of questions can emerge. In this section, we'll take a look at several different sources of raw material and consider what kinds of questions might emerge from them. This won't necessarily be an exhaustive list of all possible sources and types of questions but should provide plenty of inspiration.

Ideas

Perhaps the most obvious of the four categories, ideas that attempt to solve a problem or hit a target are relatively natural to come up with—both within the team and, externally, from business stakeholders or passionate users of your product. You may even have previously run design workshops to generate ideas and solutions. These are all good fodder for experiments but don't fall into the trap of automatically putting forth an idea as hypothesis for an experiment.

Instead, take the ideas and think about the premises that might underlie them. By this, I mean that each idea is likely to have several underlying beliefs which presume that if they're employed, a positive change will result. But this isn't always the case.

A great example of this comes from Dan McKinley, formerly of Etsy. He tells the story of the team's belief that they could boost sales of items listed on Etsy by introducing a particular design solution—infinite scroll—on their product listing pages. After all, if people saw more items with less effort (i.e., in a shorter space of time and less clicks), more items would be seen and thus bought.[4]

The folk at Etsy tried this—they built infinite scroll functionality, fixed the inevitable bugs, and then A/B tested to see whether their hypothesis was correct. It wasn't. Users who were in the "infinite scroll" experience actually ended up seeing, and buying, less, through the infinite scroll search mode.

After some time of disbelief and hand-wringing, they took a step back and decided to break down their hypothesis into two, smaller **premises**, phrased as questions.

"Are more items in search results better?"—this, they could test just by changing the number of items loaded onto a normal search result page. The result—yes, but not by much. Certainly not enough to suggest an overwhelming benefit to increasing the number of search results shown.

[4] McKinley tells this story, and offers other great advice, in his presentation "Design for Continuous Experimentation," available at https://mcfunley.com/continuous-experimentation/

"Are faster results better?"—this, they could test by slowing down the speed of fetching results on search. The result—no difference in number of products bought.

Both of these questions were much simpler to answer—they could be answered quickly and easily with some simple configuration changes. What's important to note here, though, is that rather than experimenting on an idea, they tested the underlying premises.

■ **Remember!** Avoid using experimentation to test design solutions, at least until you've identified and tested the premises that underlie them.

Claims

Like it or loathe it, regardless of whether you work on an established or totally new product, people are going to make claims about your work. Marketing colleagues will try to identify persuasive lines to help sell the product. Others in your organization will have an idea in their minds of what the benefits and shortcomings of the product are. And of course your users will frame their own perceptions and opinions.

Although all of these are often ignored as a source of material for questions and experiments, they can be very interesting and influential to investigate. For the positive claims, you might want to question whether these claims are actually persuasive to your core audience—if they are, this might lead to new idea-based questions around how you might best demonstrate these benefits. For negative claims, you can choose to question whether these claims are actually supported by evidence, in an effort to reshape people's perceptions.

Assumptions

We all make assumptions in our day-to-day as well as professional lives. Some of these are harmless, and if we stopped to question every single basic assumption we make, we'd never create anything new. However, just because it's a piece of accepted wisdom doesn't mean that the world, or your users, haven't moved on by now.

Sometimes, these might be pieces of industry knowledge. An example from our team's experience. Our recommendation pop-up was doing pretty well, but we wondered how we might improve the click-through rate. Conventional wisdom suggested that including an image next to the item we were recommending might improve things—images were considered more appealing and enticing than just text.

So we tried it. We built something quickly, and—nothing happened. If anything, those with images performed slightly worse. In the context that we were working with, this particular piece of conventional wisdom didn't seem to apply.[5]

Other times, it might be the case that times, or our users, have changed—what we discovered from one experiment, months or even years ago, may not ring true today. What works for one set of users may not work for others. Although we can design experiments to control as many factors as possible, it's best to never assume, regardless of the results of an experiment, that the answer you uncovered once will stand for all time and for all users.

Assumptions can range from accepted industry wisdom to assumptions about the demographics, behavior, and beliefs of our audience. If there's an assumption we have, we need to test it through experimentation—through asking a question and establishing whether or not the assumption holds up in the face of evidence.

It's worth noting that assumptions and the premises that underlie ideas are closely related. I prefer to think of assumptions as things like industry wisdom and our beliefs about the audience or our product rather than assumptions that a particular solution will lead to a positive effect. Those, to me, are premises, and can be much more difficult to bring to the surface—but it's worth it.

Knowledge gaps

Sometimes, we have to own up to the fact that there are things we don't know.[6] With ideas, claims, and assumptions, we don't necessarily *know* they are useful, true, or false, but we at least have a hunch, or prior belief, that they may be.

Knowledge gaps are the areas where you really don't know at all. You may believe there's something interesting out there waiting to be discovered, but what that is, you couldn't possibly say. They arise when we have to put our hands up and say—"I just don't know, but I'd love to find out…."

This is often where a particular form of qualitative research comes into play. Sometimes referred to as "interpretivist," "abductive," or "relativist," these kinds of questions aren't necessarily answered by establishing whether something is true or false.

---

[5] At least, not to the extent that it was worth investing more of our team's effort in.

[6] The Johari window is a framework you may have heard of—known knowns, known unknowns, unknown knowns, and unknown unknowns. Here, we're talking about the known unknowns—things we know that we don't know—identifiable gaps in our knowledge.

Instead, researchers using this technique observe the context within which users of your product go about their lives—how and why they interpret the world around them in their own ways. The answers to the questions posed by these kinds of experiments are often answered by new premises, claims, and assumptions, taken from direct observation.

Of course, identifying a knowledge gap can presuppose that there *is* knowledge to be gained. For instance, you may want to find out who uses a particular feature of your product, how, or why they use it.

There's an underlying premise here, though, that people use the feature in the first place, that there's an identifiable reason why, or coherent strategy for how they use it. Which may or may not be the case. All of which can lead to a number of interesting questions—some more focused on determining whether there is knowledge to be found and others digging deeper into that knowledge itself.

Moving from raw material to experiment-ready questions

By now, you should have a healthy stack of index cards, sticky notes, or whatever you could get your hands on, full of raw material for questions. Next, we need to reexamine each of these pieces of raw material and turn them into a set of experiment-ready questions.

What are "experiment-ready" questions?

Experiment-ready questions are, quite simply, questions that can be answered by running an experiment, that is, by doing a specific piece of work in a reasonable amount of time, by one or many members of a multidisciplinary product team.

Generally, these tend to come in two flavors:

- Belief-led questions, where you have a prior belief, premise, or assumption that you want to test against reality
- Exploratory questions which seek to gather knowledge where you have no existing beliefs, premises, or assumptions

There is sometimes an assumption within product and design communities that any question that seeks to understand "why" or "how" cannot have a hypothesis behind it.

This is then extended to the point that these kinds of questions cannot be answered by quantitative methods (the collection of numerical data) and must only be answered by qualitative methods (the collection of descriptive evidence).

This is similar to the fallacy that all experiments are A/B tests, and every experiment must be data (i.e., numerically) driven. However, an experiment is just a structured way of asking and answering a question. It's perfectly possible to have a prior belief, assumption, or claim about how or why something happens and thus to answer those questions in a belief-led manner.

The notion of pitting "qual" against "quant" research is a fruitless battle. Both approaches have strengths and weaknesses, and as ever, a combination of both, known as "mixed method research," can get the best of both worlds.[7] The important thing to bear in mind here, though, is that you should always start with the question to be answered. Design your experiment as best you can so that you can get a useful, meaningful answer, and *then* select the most appropriate method to gather that data, numerical or descriptive.

Remember! Don't fall into the trap of arguing over the merits of "qual" vs. "quant"—concentrate on the question, and whether it's hypothesis-led or open-ended. Selecting a method to answer the question is a completely separate concern, regardless of the question's flavor.

Belief-led questions

The former are the traditional domain of "experiments." These are questions that work very well with a mindset of seeking to test, or falsify, a belief, in accordance with the philosophy of science made famous by Karl Popper in his book *The Logic of Scientific Discovery*.[8]

Belief-led questions can themselves be broken down into two general types:

- Questions that test premises of what we believe the state of the world to be (or has been at a specific point in time). These include questions that test a belief about how users behave or have done so in the past.
- Questions that test premises of interventions, or changes we could make, that we believe will lead to positive effects: *"Would doing X lead to change Y?"*

---

[7] For a much more in-depth discussion on mixed methods, I'd recommend reading *Mixed Methods: A short guide to applied mixed methods research* by Dr. Sam Ladner, http://mixedmethodsguide.com

[8] Karl Popper, *The Logic of Scientific Discovery*, 1959, Routledge

Regardless of the type, however, a belief-led question is ultimately trying to ascertain whether something is true or false—whether the team is right or wrong to believe in something specific.

Exploratory questions

Exploratory questions, on the other hand, tend to focus on what we've been calling knowledge gaps—areas where you have no prior belief, other than there being something to fill that gap, that is.

As mentioned earlier, it's often worth testing *that* belief first before rushing too quickly into an experiment that seeks to fill the gap, but more often than not, these questions can be hugely valuable for simply understanding the context around your users. This in turn can inform much better belief-led questions.

For exploratory questions, it's still very important to design the experiment—to think carefully about how you'll gather and measure data, who you'll want to include as a participant in the research, and what you can do to ensure the results are useful and meaningful.

A note on determining the scale of an experiment

In Chapter 7, we'll look at several concepts which can be used to determine the scale of an experiment. Scale refers to the amount of data you need to capture, in order for the answer to be a useful, meaningful one.

These concepts are extremely important to bear in mind, particularly when it comes to belief-led questions. Exploratory questions may not use the specific statistical techniques that we'll look at in Chapter 7, but the concepts that they address are still worth bearing in mind.

The Five Ws (and one H)

Once you've gathered your raw material as a team, a number of questions will naturally arise. These can be things that the team is really keen to understand or know, or something you feel the team or the business really needs to know before any decisions are made.

Sometimes, however, it can be hard to come up with questions that aren't just a rewording of a premise or assumption. There's nothing wrong with those kinds of questions, but they can be quite limiting.

A worthwhile exercise at this stage in the process is to try using the "Five Ws." This is a well-established technique, used throughout history to try and understand the basic facts of any situation.

The Five Ws (and one H) are

- Who
- What
- Where
- When
- Why
- How

In the context of preparing experiment-ready questions, these can be used in two ways. Firstly, they can be used to flush out any extra premises or assumptions that might have remained hidden in previous discussions. For example:

- *Who do we believe uses this feature?*
- *When do we think people use it?*
- *Where do we think people are going to use solution X?*

By posing these questions to the team, you can uncover more premises and assumptions that lurk behind either the team's view of how things stand right now or are implicit in proposed design solutions. These premises and assumptions can then be tested via belief-led questions.

Secondly, you can use any of the Five Ws (and one H) to create exploratory questions that seek to understand the current state of play in the world, where you have no prior beliefs or knowledge.

Prioritizing your backlog of questions

You could spend forever asking questions, using the experimental way of thinking to answer them, and not get very far. What's crucial is to make sure that you're tackling the important, useful questions—the ones that are most valuable to know the answers to.

Of course, you won't always know which questions are valuable up front. Indeed, one of the greatest pleasures of experiment-driven product development can be taking a seemingly innocuous question and discovering something wonderful and game-changing. Still, it's important, early on, to consider and make explicit the possible value behind answering each question you have.

There are two ways we can do this—by **asking why** and by using a simple scoring mechanism to **rate the potential value** of answering each question.

Asking "why?"

Capturing the motivation, the reason for asking, and more importantly, wanting to answer, a question, is vital when determining its priority. It's only by asking and capturing "why" that you can make it clear among the team and to stakeholders, the value behind spending effort in running an experiment.

Clarity among the team harks back to one of the key principles I outlined in Chapter 3. In the day-to-day work of a team, it's sometimes very easy to lose sight of why a piece of work is being done. Capturing what value an answer may bring, and having that as visible as possible, helps keep a team motivated. It also, in combination with capturing the question, really focuses the mind when trying to determine what the simplest, useful thing might be to answer the question.

Keeping focused on the question, and why you're seeking an answer, also helps ensure that you don't get distracted by any one solution or thing you're doing to answer it. If it's not the best way to answer the question, remember why you're asking it in the first place, and rethink what you're going to do to get there.

Secondly, it's a great communication tool with stakeholders outside of the team. For instance, at product demos and showcases, we commonly run through not just the work we're doing, or results of experiments we've run, but upcoming experiments too. Stating the reason why we're doing particular pieces of work, or proposing to run certain experiments, helps stakeholders understand, and build trust in, the work you're doing.

How valuable might an answer be?

In addition to asking, and capturing, why we want to answer a given question, you should take the time with your team to get a sense of the value of answering a question. Rather than scoring the complexity of whatever work it would take to answer the question, or simply getting people to vote on which question they "like" the most, I believe there are four criteria which can help you assess the potential value of an answer.

- **Confidence**: How certain is the team that we already know the correct answer to this question?
- **Risk**: What's the danger if we're wrong about the premise, or if we don't answer the question soon?
- **Impact**: If we know the answer, and/or we're right about the premise, how much benefit might that bring to the team or to our users?
- **Interest**: What appetite is there within the team for finding an answer to this question?

■ **Note** While discussing the "risk" and "impact" criteria with the team, you're likely to uncover the reasons **why** you want to run the experiment in the first place.

One way of prioritizing among questions would therefore be to score each question against these four criteria.[9]

For each of the criteria, you'll come up with a number—then you'll add the various ratings together to get a final score. The exact scale you choose to use is less important than the general principle that a **higher number** should equal **more importance** in this system.

Add up each of the ratings, and you'll have a final score for each question. In the event of a tie, use your judgment—but a good rule of thumb is to err on the side of prioritizing the question that scored higher on the 'risk' scale.

Let's look at an example in Figure 5-3.

Question	Confidence	Risk	Impact	Interest	Total Score
Question A	2	4	3	5	**14**
Question B	3	4	4	4	**15**
Question C	4	2	4	4	**14**
Question D	5	2	3	3	**13**

Figure 5-3. An example of a question prioritization matrix

[9] As always, no method is ever going to be perfect. Use the scoring as a guide, but use your judgment too.

As you can see, we have a clear winner—Question B—where we are pretty confident we know the correct answer, but the risk if we're wrong is still large. Ultimately, it's the confidence that we have, which has swung it. Often, you can be confident in an answer, and be surprised when you're wrong. If you're right, that's a morale-boosting win for the team. If you're wrong, well, at least you were wrong quickly and cheaply.

We have a tie for second place—but one has a higher score for risk, so we should prioritize Question A above Question C. By coincidence, Question A also has a higher level of interest from the team. While being interested in finding an answer is certainly a huge factor, you should avoid making the interests of the team the sole driver for prioritization.

Remember! In the event of a tie for highest score, prioritize the question with the higher risk.

Congratulations, you have a first cut of a prioritized backlog of questions!

After running the previously discussed exercises, you should come away with a prioritized backlog of questions to experiment around. It's likely that we could classify this backlog in one of two ways:

- A *broad* backlog is one that is composed of questions that seek to probe lots of different areas of your product space.
- A *deep* backlog is one that sticks to one or a very few areas of the product space but instead imply a chain of related questions—if question A has result B, then do question C, else do question D, and so on.

As a general rule, it's better to start off with a broad backlog. Mainly because if you begin with a deep one, and question A results in proving a vital hypothesis wrong, that could render most, if not all, of the backlog redundant.

That said, as you begin investigating a broad backlog of questions, by its very nature, each experiment is likely to inspire new questions. Over time, each of the original questions on the broad backlog could have their own deep backlogs. But not all of them will. And if you can knock out the ones which aren't helpful, as soon as possible, that's great.

Of course, you don't know that it's not helpful beforehand, but the prioritization exercise earlier should help give some indication of that. Ultimately, it's up to you as a team to decide when to move on from investigating one deep

set of questions to another branch of the broader backlog tree—it's worth running frequent exercises in prioritization to help you work out when the switching point makes sense.

Bringing it all together—The experiment card

The experiment card is the key artefact that underpins the process of experiment-driven product development. It's a format, or structure, that gives you the key pieces of information you need to nail down before running an experiment. These can be captured using both sides of an index card, which can then be used in the context of a progress monitoring tool such as a Kanban board.

The experiment card is also something that you can use for communication with those outside the team. A clear template that fits neatly on a slide for presentation, you can use experiment cards as a way to frame the work that you're doing, at demos, showcases, sharebacks, and so on.

What should you capture on an experiment card?

The experiment card needs to communicate the vital pieces of information that the team, and stakeholders, will need, in order to understand the work being done. See Figure 5-4.

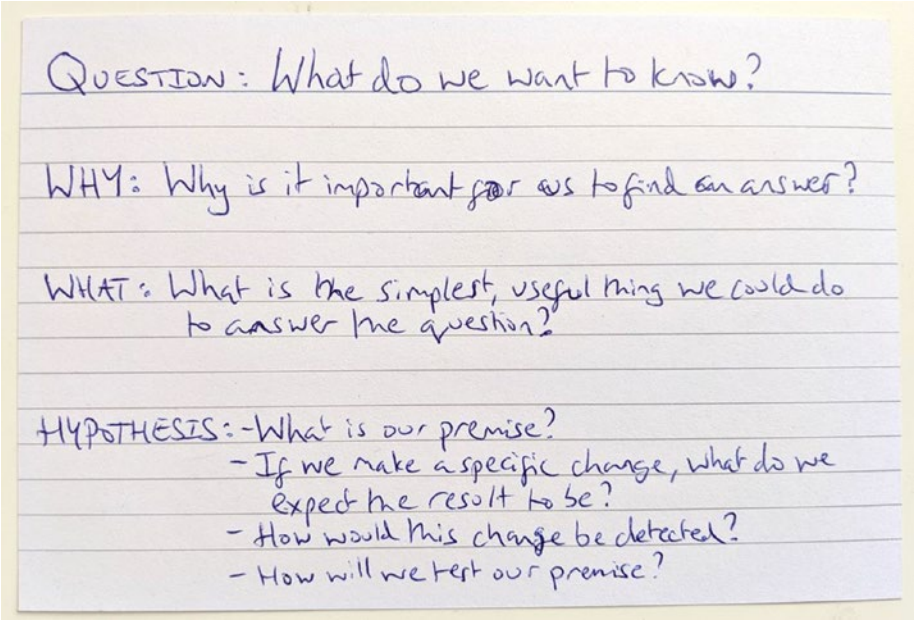

Figure 5-4. The front side of an experiment card template for hypothesis-led questions

On the front side of a physical experiment card, we write down

- The **Question** we want to answer
- The reasons **Why** we want to find an answer
- **What** the simplest, useful thing is that we will do to answer the question
- And our **Hypothesis**, if applicable—essentially our prior belief or premise

In this chapter, we've discussed the importance of the Question, and communicating the reasons behind seeking an answer. In Chapter 6, we'll go into more detail about the hypothesis.

Note that although we talked about the *principle* of Simplest, Useful Thing in Chapter 4, we didn't directly discuss the method, or what you will specifically do to answer a particular question.

Although the "What" is specified up front here, it's important to reiterate that **you should not decide the method** until you have discussed and decided the other items on the experiment card.

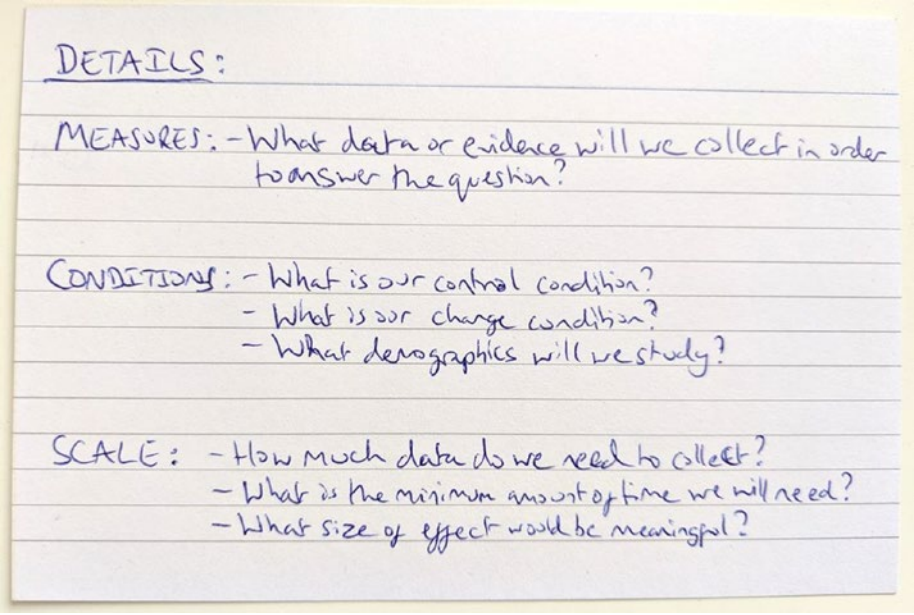

Figure 5-5. The reverse side of an experiment card template, capturing the finer details of the experiment's configuration

On the reverse side of the physical card, we go into more detail about how exactly we are going to design our experiment. See Figure 5-5. Here, you should write down

- The **Measures** we will use to answer the question
- The **Conditions** under which the experiment will be run
- The **Scale** of the experiment—how much data we need to collect in order for the answer to be meaningful and useful

As you can see from the template, there are a number of additional questions under each heading, which are useful to ask when discussing and deciding. In Chapter 6, we'll cover the measures and conditions, and Chapter 7 will be all about the scale of the experiment.

Summary

In this chapter, we've started to get into the real detail of practicing experiment-driven product development. Here, we've covered **the Planning Phase**.

This is the phase in which you begin to gather together all the possible things you could experiment with, and work out what to tackle first.

We've covered

- Various ways to find **inspiration** for experiment-ready questions
  - **Ideas** for solutions to business and customer problems
  - **Claims** that we or others make, both good or bad, about the product
  - **Assumptions** we have about our product, our users, our industry, or our world
  - **Knowledge gaps**—areas where we have no preconceptions, other than that there is a gap in our knowledge that we would benefit from filling
- The importance of mining this raw material for **premises** that lurk in the background—these are the true fuel for our experiments

- How to move from the raw material to experiment-ready questions
    - The difference between **belief-led** and **exploratory** questions
    - The importance of not getting drawn into the "qual" vs. "quant" debate
    - **The Five Ws** (and one H) technique and how it can help inspire more questions
- Ways in which you can **prioritize** the questions you want to answer
    - Capturing the **"why"**—the value we think an answer might give
    - Using a scoring system to evaluate the **confidence, risk, impact, and interest** in a question

Finally, we started digging into the **Experiment Card**, which will form the backdrop for the rest of the book. With this all in mind, then, we can move out of the Planning Phase and begin **designing** an experiment.

In the next chapter, we'll cover the next section of the experiment card—namely, formulating a **hypothesis**, choosing appropriate **measures**, and getting into more detail about the exact **conditions** of the experiment.

CHAPTER

6

Hypotheses, Measures, and Conditions

In the previous chapter, we looked at what is known as the "Planning Phase" of experiment-driven product development. In that phase, we collected together all kinds of raw material that form the basis of product development work, turned them into experiment-ready questions, and prioritized among them.

Once complete, you'll have a backlog of questions that if answered would give you useful knowledge that can help shape your path forward. Now, we're ready to pick a question from the top of the backlog and move into the "Design Phase."

The Design Phase of experiment-driven product development is crucial to the success or failure of any particular experiment. In the Design Phase, we take the question we're going to try and answer and design an experiment which will yield a useful, reliable answer—one that we can trust to make a product decision upon. This will be the focus of Chapters 6 and 7.

P. Rissen, *Experiment-Driven Product Development*,
https://doi.org/10.1007/978-1-4842-5528-5_6

Allowing the time for experiment design is a crucial part of the XDPD approach. It's very easy to dive straight into running experiments, particularly if you're used to A/B feature testing, but until you've become comfortable with nailing down exactly what you're trying to find out, the precise way in which you can discover that, and interpreting the results of your experiment, you'll find the whole experience somewhat unsatisfying—and it won't really help improve your product or service.

In this chapter, we'll concentrate on the first part of designing an experiment—establishing a **Hypothesis** for your belief-led questions, choosing appropriate **Measures**, and detailing the specific **Conditions** needed in order to answer the question. In the following chapter, we can then discuss the **Scale** of the experiment, and the **Method** you're going to use.

In Chapter 5, we discussed two common forms of question that emerge from the Planning Phase of XDPD—"belief-led" questions and "exploratory" questions. As you can see from Figure 6-1, no matter what form of question you're dealing with, the Design Phase has plenty to offer.

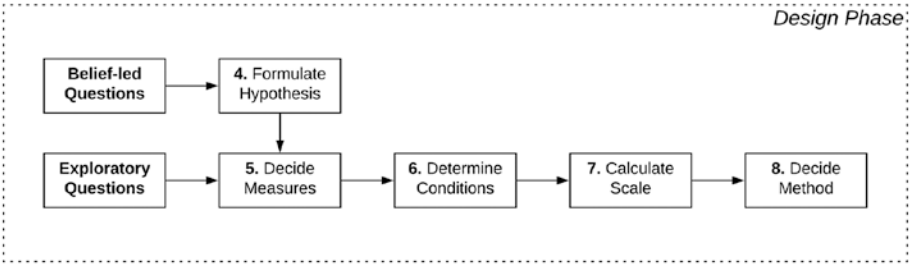

Figure 6-1. The Design Phase of experiment-driven product development

When considering belief-led questions, however, there is an important step to go through before you can get down to deciding measures, conditions, scale, and method—formulating a hypothesis.

Formulating hypotheses

A hypothesis is a statement of belief, hence why it is a mandatory requirement when trying to answer belief-led questions. After all, if you're trying to determine whether your belief is correct or not, you need to start off by being specific about what exactly it is you believe, so that the experiment can actually provide a useful answer.

In digital product and user experience circles, hypotheses are often framed around solutions. That is to say, the team or stakeholders are asked to prepare a *hypothesis statement* which frames a solution idea in terms of how it

meets the particular needs of a group of users and how a product team will know whether or not the idea was a success. Often, the talk that follows is of how the hypothesis can be "validated" or, even more confusingly, "proved."

In XDPD, we follow the established practice of scientific and statistical experimentation—we talk of **testing** a hypothesis rather than validating or proving one.

The reason for this slight change is subtle, but it can help you greatly when designing experiments. The danger in framing hypotheses as things to be "validated" is that it can lead people down the path of designing experiments that are biased in favor the collection of positive, affirmatory evidence.

Similarly, claiming that an experiment which affirmed your belief in something, "proved" it, suggests that the matter is closed forever and can never be disputed. As emphasized throughout the first few chapters of this book, we need to recognize the limits of our conclusions, and stay humble.

Remember! Instead of validating, or proving a hypothesis to be "true," we seek to test a hypothesis and only accept it if it meets acceptable levels of certainty.

So, what makes a good hypothesis? Let's take a look at a few key principles of good hypothesis design and some examples of how you can use them to write good and poor hypotheses.

A hypothesis should be the presumed outcome of your research

A good place to start when designing a hypothesis is to start with what you believe the answer to the question to be. Having started with a premise, claim, or assumption and turned that into a question, you probably already have some inkling, a hunch, or opinion perhaps, of what the answer might be. Whether that answer is right or wrong doesn't matter at this stage. What is important now is to capture what you *think* the answer might be.

This is important because it forces you to be honest with yourself, acknowledging that you have opinions and biases and putting them out there in the open, before the experiment begins. In some ways, it's the equivalent of declaring any potential conflicts of interest before you begin in earnest.[1]

---

[1] It also helps to avoid what's known as "HARKing," or "hypothesizing after the results are known." Sure, when analyzing the results of an experiment, you can come up with *new* assumptions, premises, and so on, but these are not equivalent to hypotheses that will actually yield useful answers.

If it turns out you were wrong, OK, you might look silly for a moment, but this is where doing the smallest, useful thing comes in. Spending a little effort to prevent you from wasting further time and resources pursuing a lost cause in the future can be a price worth paying.

A hypothesis must be falsifiable, testable, and measurable

Making claims is easy. When we are choosing what to believe, in order to make good product decisions, however, we normally need a higher standard of proof than gut feeling. The three criteria—**falsifiable, testable, and measurable—**are necessary to achieve that standard.

Falsifiable

What do we mean by falsifiable? A hypothesis that is not falsifiable is essentially impossible to refute. There's no feasible way of really knowing whether it can be trusted or not, so there's no purpose in trying to answer the question it poses.

In contrast, a falsifiable hypothesis is one in which it is conceivably possible that we might find evidence that disproves it. We might never be able to emphatically *prove* a hypothesis—because there's always a chance that in the future it may no longer be true—but we *can* disprove one, by collecting evidence which shows it not to be true.

Testable

Along similar lines, a hypothesis needs to be testable. If there's no practical way in which we can reasonably assess, given the time and resources available to us, whether a hypothesis stands up to scrutiny or not, then it's useless to us.

For example, in the world of scientific publishing, we might hypothesize that building a particular feature might lead to lesser-known research papers being cited more often than otherwise. But the pace at which research papers are written, reviewed, and published means that we'd only be able to collect evidence to test that hypothesis after months, if not years. More often than not, working on digital products, we just don't have that luxury.

Measurable

Every experiment relies on evidence; thus every hypothesis needs to explicitly state what measures we will use when assessing whether to accept it or not. Most often, this takes the form of some kind of number, but that doesn't mean that every experiment must use quantitative methods.

The criteria of measurability simply means that there must be some way of comparing the evidence gathered during the experiment back to the original hypothesis. A hypothesis should always state the specific measure that it can be assessed using.

Furthermore, building on the principle that your hypothesis should be the presumed outcome, or answer, to your research, you should state in the hypothesis *what you believe will be found* when examining the evidence.

The raw material from which we created questions can be quite general—*we think adding a photo feature will increase engagement*—but it's difficult to accurately test whether this is true or not without defining the conditions in which someone might see the photo feature, how exactly you would measure "engagement," or to what extent it might be increased.

A hypothesis often describes a difference or change

When considering your raw material—the premises, claims, assumptions, and knowledge gaps—it's often the case that at their heart, each of them conceives of a potential difference or change that could exist or will result once your work is complete.

This could be a positive or negative change to a product metric, a change in behavior or opinion, or simply a difference in behavior between two groups of users. A good rule of thumb, therefore, when formulating a hypothesis, is to think about the kind of difference or change it represents.

This gets quite philosophical and deep, but think about it this way. If there were no distinctions between any knowledge, or indeed between the users of your product—if all there was, was an amorphous mass, where nothing changed and there were no meaningful differences in the contexts, beliefs, actions, state, or thoughts of our users, then how would you develop anything? How would you be able to pinpoint a particular thing that made the product better? How would you be able to distinguish between different user groups? It'd be impossible.

Innovation, as Jared Spool has written,[2] isn't necessarily coming up with a shiny new thing—it's making a positive difference—affecting a change, delivering something that satisfies and delights someone, because it makes their lives unexpectedly easier.

Describing precisely what that change, or difference, might be is important. It helps make the hypothesis testable, because it tells you exactly what to look out for once you've gathered the data from running an experiment.

Gathering evidence via running experiments, no matter what form that might be—user activity data, interview records, and so on—and then analyzing the evidence to determine whether the change or difference you thought might be detectable, is there or not, will ultimately provide the knowledge you sought. It might not be the answer you expected, but it will be useful knowledge, to inform what comes next.

Regardless of the method you choose, defining your hypothesis requires that you think carefully about what difference or change you're trying to detect. It could be quantifiable or purely qualitative.

The hypothesis statement

Now we've looked at the principles which make up a well-designed hypothesis, let's look at the general format of the *hypothesis statement*—the sentence we will write on our experiment card and communicate to others.

At an absolute minimum, the hypothesis statement should contain three things:

> *We believe the answer is <our belief>,*
>
> *The evidence for which would be reflected in <our measures>,*
>
> *Under the circumstances of <our conditions>.*

Notice that the hypothesis statement:

- Is framed as an answer, X, to the question being posed
- Clearly states the way in which we expect to be able to detect evidence (i.e., our measures)
- Also states the conditions under which we expect the answer to hold true

[2] "What Makes an Experience Seem Innovative?" 2013 https://articles.uie.com/innovative_experience/

Our work here isn't complete, however. Although a hypothesis constructed according to the preceding template would be measurable, it could still lead us to design an experiment weighted in favor of our belief. To avoid this bias, and to ensure we stay honest in our experiment design, we should pair our hypothesis with a *null hypothesis*.

Null hypotheses

Hypotheses always come in pairs. When we talk of "testing" a hypothesis, we're not just directly trying to gather evidence which suggests this hypothesis may be correct or not. Instead, we're trying to decide whether to reject what's known as the *null hypothesis* in favor of our *alternative* hypothesis.

The null hypothesis is essentially the mirror to your hypothesis. Null hypotheses generally tend to state that

- Nothing will happen as a result of a change you make
- Anything that does happen, or any change/difference you notice, is purely down to chance
- The reason you believed was responsible for a certain phenomenon is not correct
- There is no real answer to the question

This may sound cynical, but in order to make sure we don't mistakenly end up accepting our alternative hypothesis, and thus making poor decisions, we must presume, up until we have assessed the evidence gathered from an experiment, that the **null hypothesis holds true**.

What we actually do in an experiment to answer a belief-led question, therefore, is work out whether the evidence we gather is strong enough to persuade us to *reject* the null hypothesis in favor of our proposed alternative answer.

Examples of hypotheses

Let's run through three examples so that we can understand the concept of our paired hypotheses—the "null" and "alternative" hypotheses in more detail.

Example 1

Question: Is it true that changing the color of a "buy" button would increase sales in China?

Belief: Red is regarded as a "lucky" color in China; therefore we should make more use of it for that audience.

Hypothesis: Changing the "buy" button on product pages in our site from blue to red will increase sales in China by 3%.

Null hypothesis: Changing the color of the button will not be responsible for sales in China increasing by 3%.

Note that the null hypothesis doesn't say that sales won't increase at all. What it says is that *changing the color of the button from blue to red will not directly cause sales in China to increase by 3%*. Sales might increase. They might decrease. They may stay exactly the same. All of this might happen by chance.

The key point here is that the null hypothesis asserts that the relationship between the change in color of the button and the precise jump in sales doesn't stand up to scrutiny.

Example 2

Question: Would anyone care if we removed Feature X from our product?

Belief: Given that we can't see any evidence of click data in the past 12 months on Feature X in our product, we should remove it.

Hypothesis: Nobody has clicked on Feature X in the last 12 months; therefore it is unloved and can be removed.

Null hypothesis: Users have noticed and appreciated Feature X over the last 12 months, but this is reflected in ways other than click data.

In this example, you'll note that we're very specific about the measure—click data—and the conditions—the last 12 months. Our null hypothesis helps frame our experiment so that we can get a better answer, by encouraging us to look at other possible evidence before we take a potentially rash decision.

Example 3

Question: Who are becoming our most engaged user group?

Belief: Students in Argentina are becoming more engaged in our product than any other user group.

Hypothesis: We believe students in Argentina are becoming our most engaged user group, as evidenced by the fact that they visit more pages per session than any other group.

Null hypothesis: Students in Argentina are no more engaged than any other user group—the increased number of pages per session from that group is a random event.

Again, even though this example doesn't propose an intervention or change to the product, we are careful to state both the measure—pages per session—and the conditions—students in Argentina. The null hypothesis cautions us against reading too much into the data we've collected, by reminding us that chance is always a factor.

■ **Remember!** Every belief-led question needs both a hypothesis and a null hypothesis. Your experiment is then designed to determine whether there is enough evidence to reject the null hypothesis and accept your "alternative" hypothesis.

Exploratory questions and hypotheses

At first, it might be presumed that hypotheses are mandatory for all experiments. Indeed, if you think of experiments purely as A/B tests (and/or as trials of new features or changes to an existing product), then this makes sense.

However, in XDPD, experiments are a structured way of asking and answering questions. The method, as I've said previously, should only be decided once you know exactly what question you're trying to answer, and the simplest, useful way of answering it.

When considering exploratory questions, you might at first find that coming up with a hypothesis does seem possible. In this case, you're likely to have encountered a lurking belief—or worse, the method-question-answering loop.

The lurking belief

Unknowingly, you've actually surfaced a premise, assumption, or claim—some form of prior belief as to the answer to the question. That is fine, but this turns your question in to a **belief-led** one, and thus a hypothesis is needed.

The method-question-answering loop

Your proposed hypothesis is either very general—"if I'll do research, I'll find an answer"—or too specific to a particular method—"I believe that if we do a survey, we'll find an answer." This is what I call the "method-questioning-answering" loop.

The method-questioning-answering loop is one where because you've baked a method into your hypothesis, you end up designing an experiment to test whether that particular method would answer the question or not.

That means ultimately the question you're asking *isn't* the original intended question. For example, let's take an exploratory question such as *"Why do people think our product is difficult to use?",* and let's assume that you have no prior knowledge, assumptions, or beliefs, so you propose it as an exploratory question.

When trying to come up with a hypothesis, you end up stating a belief (already a warning sign when dealing with exploratory questions) along the lines of *"We believe if we run a survey, we'll be able to answer the question."*

Remember that your hypothesis should be framed as an answer to the question. If we were to continue along the lines of the belief we just outlined, then we'd not be answering "Why do people think our product is difficult to use?" Instead, we'd be answering the question of *"Would a survey tell us why people think our product is difficult to use?"*

Perhaps it would—but if it didn't, you'd end up looping back around and proposing a different method, perhaps some user interviews. Either way, you've not actually discovered the answer to the question you posed in the first place.

Again, this is where our null hypothesis can save us. If you find yourself falling into the trap of the method-question-answering loop, try to come up with a null hypothesis.

While yes, your method-based hypothesis *is* technically a hypothesis, your null hypothesis would be something like "if I don't do research, I'll not find anything" or "A survey won't answer the question," which isn't very helpful.

To be safe, don't force a hypothesis into your experiment which is answering an exploratory question. And if you spot yourself doing so, take a step back, think about whether there's really a lurking belief here—and avoid falling into the method-question-answering loop.

■ **Remember!** Although hypotheses may not be mandatory when it comes to experiment-driven product development, this doesn't mean you can skip the Design Phase altogether! Considering conditions, measures, scale, and method is still crucial to the success or otherwise of your experiments.

Having formulated our pair of hypotheses, our belief-led questions can now return to the main track of the Design Phase. I've touched upon the importance of making experiments measurable, and specifying the conditions under which we expect to find evidence, but let's close out this chapter by digging deeper into these topics.

Measures and conditions

Every experiment starts with a question. Whether or not you have a prior belief as to the answer, by designing an experiment, you'll want to find ways in which you can gather evidence which will help you answer the question.

One way in which we can do this is to think about **how** we expect evidence to manifest in the world and what the **motivating factor** might be in causing that evidence to appear. These correspond to our **measures** and **conditions**.

Measures

Running an experiment without deciding which measures to pay attention to can leave you stumbling around in the dark when it comes to analyzing the results and determining an answer.

We select measures so that we can ensure we capture evidence which will help us answer the question but also to help us catch any other interesting secondary evidence—things that might inform future experiments, or things that make sure we're not damaging the long-term health of our product or service.

You need to be able to measure at least one thing, in order to be able to answer the question—to see, for instance, whether you were able to detect a change or difference, and how small or vast that change was. This will be, ultimately, what determines whether you can reject your null hypothesis in favor of an alternative.

Even in the case of exploratory questions, where you have no prior belief or hypothesis to work with, specifying the measures you're especially looking out for, ahead of time, will ensure that your research is targeted.

This should never be to the *exclusion* of other signals or material that may surface. When exploring, you should always be open to new, interesting things. But by writing down the measures ahead of time, you ensure everyone involved in the experiment, and your stakeholders, know what kinds of evidence will help answer the question.

Choosing appropriate measures

When deciding which measures to choose, think about **how you would expect your answer to manifest in a way that is possible to detect**.

Perhaps people will click on something in particular. Perhaps they'll spend less time on a certain step in a process. Perhaps they'll smile more or drop a particular word in conversation. Select the thing that, if measured in some way, will be representative of the change or difference or would help you answer the question when you have no prior belief.

Measures can appear simple at first—number of clicks, number of page views. But bear in mind that you want to also make sure that they are **representative**. For instance, if measuring clicks, if you only measure the basic number of page views, that could include a single user refreshing the page many, many times. Perhaps in that case it might be better to measure unique page views—the number of different pages that were viewed.

Measures can be **calculated**. For instance, if you want to measure the interest in something, you might want to measure the number of clicks. But sheer number of clicks could fall prey to a similar problem as earlier. In this case, the number of clicks, divided by the number of unique page views, might be more informative—the unique click-through rate.

Measures might also be **proportional**. Say you wanted to encourage users to read more articles. Measuring click-through rate would tell you that they expressed interest in articles. Measuring the number of unique page views would tell you how many different articles were read in total.

What you're really looking for is a change (or otherwise) in the number of articles read by a single user. And, more than that, what you probably really want to know is—did they spend more time reading more articles in one visit? So perhaps what you're really looking for here is the number of unique pages read in a single session by each user.

Finally, especially for belief-led questions, you should think about how you might detect whether you should be sticking with your null hypothesis—measures that might suggest your alternative hypothesis doesn't hold up to scrutiny.

These could be the same measures as those which would lead you to reject the null hypothesis in favor of your alternative. But they could also be measures which would show that, for instance, that it's not A that causes B, as per your belief, nor is it just chance—it's previously unnoticed factor C.

Track all the things, right? Wrong.

With digital products, it's very tempting to start by measuring everything, regardless or not of whether it's tied to an experiment or not. After all, if you're measuring everything, then you'll be able to learn so much more, right? Knowledge is power, and all that?

Well, not exactly. There are several downsides to simply setting up everything to be measured, whether it's in a digital or physical world.

Firstly, everything you decide to measure has a cost. In isolation, measuring a single thing, for instance, the event of someone clicking a link, may not result in running out of memory on a database somewhere, but if you measure everything, these all add up.

Similarly, every event tracked on a web site needs some kind of script to detect and fire off a record of the event, to some data collection service. The more events you track, the more those scripts can get bloated, the slower your page. Again, in small numbers, this is nothing to worry about, but it's something to bear in mind if you decide to track all the things, all the time.

Secondly, there's a lack of focus if you track everything, all the time. You're gathering all this data, and yes, it may prove useful at some point, but 90% of the time it's just sitting there, unused, useless, not really providing you with any actual knowledge. Potential knowledge, perhaps, but it's not exactly the simplest, useful thing you could do. It's the hoarder's instinct. In the spirit of Marie Kondo, ask yourself, does every single data point spark joy, or rather, does everything you're tracking *really* answer a specific question?

Finally, there are the legal, ethical, and moral implications of collecting all the data, all the time. Not only is all this data collection potentially straining your resources, and most of it sitting untapped, but it is stockpiling potential legal trouble. Increasingly, citizens and governments are, quite rightly, becoming more cautious about the amount of personal and private data being harvested and potentially sold on for profit, by both private companies and governments.

Indeed, at the time of writing, there are now laws designed to compel organizations to protect the rights of individuals, by ensuring that data is only used to provide core functionality for a product or service—and that the individual retains the right to ask an organization to provide, and possibly remove, all the information it has about their activity.[3]

All of which adds up to the indiscriminate harvesting of data being something which, while undoubtedly gives great power, also has great risks. So—if measuring everything isn't the way to go, but experiment-driven product development relies on gathering some data, where do we go from here?

Be frugal about what you measure

A rule of thumb—only measure what you need to, for the amount of time that you need to measure it. If you end up making a change to your product, or even just putting some tracking in place to measure existing behavior, only track the data for the length of the experiment. Put the measures in place, and take them out once the experiment is complete, even if the feature you developed was successful.

[3] For instance, the European GDPR laws: https://ec.europa.eu/info/law/law-topic/data-protection_en

■ **Remember!** Measures, and tracking in general, should always be linked back to a question you're seeking to answer. Determine the scale needed in order to answer it, and once your experiment is complete, remove the tracking.

Health metrics, and "do no harm"

In addition to tracking measures that can help you answer the question, it's important to make sure that, when making a change to your product or service, you're not making it worse, in unintended ways.

Earlier, we looked at an example of an experiment designed to answer whether a feature should be removed from a product. There, you'll notice that the hypotheses suggest we shouldn't just measure how we might expect to see use of the feature but other ways in which we might be able to understand whether there's enough interest in the feature to keep it around.

Health metrics work on a similar principle but are almost always independent of any particular experiment. They are the key things that you use to determine the overall health of your product or service—whether it's working or not and whether users are finding things slower, harder, or not at all.

These are measures that you'd want to know were being negatively affected, even if a particular experiment suggested that a particular course of action might improve a specific part of the product or service. Ultimately, it's up to you to make the decision, but bear in mind that in the short term, you might make things better, but keeping an eye on your health metrics will help you spot whether there could be problems in the long term.

Baseline experiments

Often, but not always, your hypotheses will suppose that if you changed something in your service or product, you'd be able to detect a difference from what happened beforehand. Sometimes, though, you'll be introducing something entirely new and won't really be able to compare it to what's gone before.

You'd still be interested to know whether there's a change or difference as a result of introducing something new—sometimes here you're especially interested in determining whether there is no difference—but directly comparing to a previous situation, or the nearest equivalent you can find in your product or service, may not be helpful.

These experiments are what we call "baseline" experiments. Rather than trying to optimize something, you're interested in initial reactions. As a result, when choosing your measures for these experiments, you're most likely to be

tracking something new and thus have nothing to compare it to. In some ways, it's less about a change and simply more about an effect. The most important thing to do in these cases is to measure your health metrics—to ensure that the completely new thing you've introduced does no significant harm to the overall health of your product or service.

Another thing to bear in mind when introducing something new is the novelty factor. We notice something new and are intrigued by it. This causes observers to fall into the trap of significantly overrating the success or otherwise of something completely new, as it's really indicating the "newness" of something rather than its actual success or otherwise.

Indeed, you almost certainly won't be able to come to significant, meaningful conclusions about the true popularity or otherwise of something completely new, in a realistic time frame of an experiment. Instead, measure your health metrics, and by all means keep an eye on measures that suggest interest, success, or failure, but do not be tempted to draw absolute conclusions the first time around. New features will often need time to bed in, a few tweaks here and there.

This shouldn't be an excuse to never kill your new features, though—never get too attached. Always frame things in terms of testing your premises, and if after a few different experiments, something isn't working as well as you were hoping, or beginning to cause sustained harm to your health metrics—kill it.

Conditions

Aside from trying to capture evidence which helps you answer the question, you'll want to know *why* this particular answer has been found. Designing an experiment around certain conditions helps you establish the **motivating factor** behind an answer.

In Chapter 5, we talked about the importance of describing why we wanted to ask and answer a question—here, we're interested in why the evidence that we uncover is what it is.

Conditions for belief-led questions

With belief-led questions, you have the pair of hypotheses, so you can determine the conditions by modeling each condition after your null and alternative hypotheses.

The condition that is designed to gather evidence to suggest you should abandon your null hypothesis in favor of the proposed alternative is called the "experimental" condition.

The "control" condition, on the other hand, is designed to disprove your alternative hypothesis—by acting as if the null hypothesis is correct.

Conditions will be familiar to anyone who's run A/B tests—the A and B being the conditions themselves. The "A" in this case is usually the control, where participants are not exposed to any change in the product, with the "B" being the group that is exposed to the new version.

Conditions are still relevant, though, in experiments which don't include direct interventions or changes to the product in question. Your conditions represent the hypothesized reasons why you believe a particular answer will be found.

For instance, you might be posing a question which seeks to discover whether or not users in certain demographics behave in a different way or have certain attitudes.

In this case, you'd want to establish a condition which groups users around the demographic that you hypothesize is a motivating factor in that difference. Equally, you'd want a condition that allows you to determine whether that difference happens in other demographics too—thus refuting your presumed reasoning.

Conditions for exploratory questions

For exploratory questions, conditions become less about "control" and "experimental" states, but focus more on the particular demographics, groupings, or categories of things you might want to study in order to answer the question.

It could be as simple as "people who use our product" or "people who don't"—but in most cases, you'll be interested in finding out more about specific targeted audiences. Thus, your conditions become **the way in which you select what qualifies something to be an object of study for your research.**

Again, look to the question and define the boundaries of what you will, and won't study, in order to determine an answer. Stating this up front helps establish the caveats and usefulness of any particular answer—always good when managing the expectations of the team and your stakeholders—but equally it helps inform any future questions you may want to ask.

Summary

We now find ourselves deep into the Design Phase of experiment-driven product development. Having chosen a question to answer, this chapter has walked you through three important aspects that will help you to design a successful experiment—the **hypothesis**, **measures**, and **conditions.**

We use hypotheses to take our belief-led questions and come up with a presumed answer, based on our prior belief. This presumed answer is often one that describes a difference or change we expect to see.

Importantly, we seek not to *prove* or *validate* that answer but to test it. We can use the concept of the null hypothesis to capture a "mirror" to our hypothesis which reminds us to look for ways in which we might be wrong about the answer.

An experiment's measures are the ways in which we will capture evidence that helps us answer the question.

For questions with hypotheses, they help us be more specific in the hypothesis, by stating exactly how we expect our presumed answer to manifest in a way we might detect.

For exploratory questions, measures help us focus our research on capturing any evidence that matters.

Conditions then give us an insight into *why* we find what we find among the evidence. They help us identify what might be the *motivating factor* behind an answer, be it a change, difference, or otherwise.

In the case of belief-led questions, ensuring a control and experimental condition helps test whether our presumed answer was correct or not. And in the case of exploratory questions, conditions help target the research by establishing the boundaries of what you will and won't study. This then sets expectations around the applicability of an answer.

With our hypotheses, measures, and conditions established, we now know what we're looking out for when running an experiment. Two other questions still remain:

- How will we know if the answer we find is meaningful enough to trust, or base a decision upon?
- After all this, how, then, do we propose to go about finding out the answer to our question?

The answer to each of these—the **scale** and **method** for our experiment—will be found in Chapter 7.

CHAPTER

7

Scale and Method

By now, you should be almost ready to get started running an experiment. In the previous chapter, we covered the first part of the Design Phase—formulating a hypothesis where applicable, identifying the measures of evidence by which we'll answer the question, and determining conditions for being part of the experiment.

In this chapter, we'll examine how we can determine the amount of evidence we should collect, in order to discover an answer that will be of any use to us. This what is meant by the **scale** of the experiment. Once this is settled, we can decide which **method** to use—what is the simplest, useful thing we could do, to gather the required evidence?

Determining scale

Scale—how much evidence we need to collect as part of an experiment—plays a crucial role in determining how useful the answer we emerge with will be.

With any experiment, there is no predetermined solution for how much evidence you need. It's up to you to decide, but there are two pivotal factors which can help you make the decision:

- How **certain** do you need the answer to be?
- How **precise** do you need the answer to be?

P. Rissen, *Experiment-Driven Product Development*,
https://doi.org/10.1007/978-1-4842-5528-5_7

Certainty here refers to how much you care about the possibility of error in any given experiment. This depends on how important you feel getting a particular answer right will be, which in turn depends on how important the decision you're likely to make as a result will be.

Precision here refers to how valuable it is to have a really granular answer. Depending on the context, moving the needle on a particular measure by 0.2%, being able to spot the difference between two very particularly demographic groups of users, and capturing as many possible varieties of responses will have different amounts of value to you, the team, the stakeholders, and the business.

How long should an experiment run for?

A question that is often asked when starting to design an experiment is—how long do we need to run the experiment for? The short answer is—for as long as it takes, in order to get a useful answer.

The long answer is—it depends. The length of an experiment should be determined by how long you're willing for it to run, and *that* should ultimately depend on how the answers to the preceding questions of certainty and precision.

When designing an experiment, you should consider how certain, and how precise, you need the answer to be. Doing this will allow you to determine how much evidence you need to collect. If it looks like you'll need an unreasonable amount of evidence in order to guarantee the certainty and precision you desire, you need to make a choice.

Both options involve a compromise on your standards of certainty and precision. One route to take would be to lower your standards in order to come up with an amount of evidence that seems reasonable and then determine the time needed to gather this new amount of evidence. The other is to set a fixed time limit for the experiment and then determine how certain and precise this will allow you to be.

In either case, compromising on your standards of certainty and precision will ensure you get *some* kind of answer in a reasonable amount of time, but it will need to be heavily caveated. By compromising on these standards, you leave yourself open to the possibility that the answer you receive is unreliable and thus cannot be used to make a product decision.

Either way, making a conscious decision to design the scale of the experiment will have wide-ranging consequences for the reliability, and thus usefulness, of the answer you end up with.

■ **Remember!** Take the time to decide how certain and how precise you need the answer from an experiment to be.

How certain do you need to be?

When it comes to belief-led questions, the experiments where we are testing our hypotheses and seeking strong enough evidence to reject our null hypothesis, there are generally four possible outcomes.

1. Evidence for our alternative hypothesis exists in reality and is detected.
2. Evidence exists in reality, but you fail to detect it.
3. Evidence *doesn't* exist in reality, and so you cannot detect it.
4. Evidence doesn't exist in reality, but you detect something that leads you to *think* it exists.

Outcome number 1 is the happy path—you ran a well-designed experiment, and the thing you hypothesized is true. Success! Well done! Give yourself a pat on the back.

Outcome number 3 is a bittersweet one. You ran a well-designed experiment, but the thing you hypothesized wasn't strong enough to dislodge the null hypothesis. Yes, it's disappointing. You'll most likely be sad, and/or go through the five stages of grief, but eventually you'll move on, having learned something valuable, and crucially, avoided wasting time going down a rabbit hole.

It's **outcomes 2 and 4** that you need to watch out for. These are the risks involved when you don't take the time to design an experiment properly.

With **outcome 2**, the danger is that after the experiment, you continue to stumble around in the darkness, assuming that the difference or change doesn't exist, when really you just failed to find it. This is known in statistics as a ***Type II error***, or a false negative. We can use a concept known as ***statistical power*** to avoid this.

With **outcome 4**, you're equally misled—you carry on thinking that you were right, when really you're (possibly dangerously) wrong. This is known in statistics as a ***Type I error***, or a false positive—mistakenly assuming that something exists, when in fact it doesn't. To avoid making Type I errors, we can make use of a concept called the ***significance level***.

■ **Note** Although these outcomes, errors, and specific statistical techniques seem particular to dealing with belief-led questions and testing hypotheses, similar phenomena exist when dealing with exploratory questions. Be aware of the ways in which poor experiment design can lead you into making the wrong decisions!

Let's take a closer look at the ways in which we can avoid false positives and false negatives. Determining the statistical power and setting a significance level are the two main ways that we can bring our required degree of certainty to an experiment in which we test a hypothesis. Therefore, these play a crucial role in calculating the necessary sample size for an experiment.

Avoid missing crucial evidence using statistical power

Statistical power is the probability that if evidence to support your alternative hypothesis exists, you'll find it. The *higher* the statistical power of an experiment, the *lower* the chances are that you'll stick with the null hypothesis when in fact there was enough evidence to reject it.

Statistical power is usually expressed as a percentage, and generally people choose a level between 80% and 95%.[1] With a statistical power of 80%, that means 20% of the evidence that would have supported your alternative hypothesis is likely to be missed. Turn the power up to 95%, and you'll only miss 5% of that evidence.

The price you pay for selecting a higher statistical power is the amount of evidence you'll need to collect. With a higher power level, you're choosing a higher degree of certainty, and thus you'll likely need to wait longer to gather enough evidence to be certain.

This is why selecting an appropriate statistical power for your experiment is a choice you have to make. You can think of it like a dial that you can turn up or down, depending on how certain of finding evidence (presuming it exists) you wish to be.

If you're running an experiment to test something really, really crucial—something that potentially could make millions, or conversely, seriously damage your brand—you probably want to be as certain as you can be, before making any kind of recommendation to stakeholders, deploying a feature, or basing design decisions on those results.

In contrast, there may be times when you're working on an experiment which, although valuable, isn't a big deal. With a lower statistical power, you're lowering your standards of evidence needed to support the alternative evidence

[1] Why not 100%? Because it's impossible to completely guarantee you'll always find the evidence.

and lowering the amount of evidence needing to be gathered overall and thus are more likely to be able to run the experiment quickly. However, this comes at a price—turn your statistical power level down to, say, 60%, and 40% of the evidence that could have been found, is likely to be missed.

Remember! The higher the level of statistical power you choose, the more certain you can be of detecting evidence to support your alternative hypothesis, where it exists. The price you pay for increased certainty—a larger amount of total evidence needed, so a longer amount of time to run the experiment.

Avoid being misled by chance with a significance level

Significance is the probability that what you detect from the evidence gathered in an experiment is down to chance. That is to say, you see an effect where in reality, it doesn't actually exist. It's always a possibility that you'll detect some evidence to support your alternative hypothesis, but sometimes this could be a freak, natural occurrence, rather than actually being a significant event.

The *lower* the significance level of an experiment, the less chance of detecting evidence supporting your alternative hypothesis which has occurred due to chance. In other words, the more stringent your standard of certainty.

Typically, people tend to set a significance level of 5%. This means that there's still a 5% chance of gathering "false positives," and doesn't *guarantee* you'll find evidence. What you're doing with a significance level is setting a certainty bar, against which you'll place the evidence you gather from the experiment.

In Chapter 8, we'll look at the concept of a *p value*, which tells us how likely any actual evidence gathered in support of our alternative hypothesis is due to chance. If the p value is less than the significance level we set, prior to running the experiment, then we can reject the null hypothesis, knowing that this evidence occurred *below* the threshold of chance that we set as acceptable and thus can be trusted.

As with the statistical power, you can choose a higher degree of certainty, in this instance by lowering the significance level. For instance, by lowering the significance level from 5% to 3%, we're saying that we want to be even more sure that the evidence supporting an alternative hypothesis gathered didn't occur due to chance.

With a 3% significance level, you're saying that you'll accept no more than a 3% probability that the evidence supporting an alternative hypothesis occurred by chance. This, as you can expect, has an impact on the amount of evidence we need to gather in total.

Remember! The lower the significance level you choose, the more certain you can be that any evidence you detect in support of your alternative hypothesis did not occur due to chance. Again, the price you pay for this increased certainty—a larger amount of total evidence needed, so a longer amount of time to run the experiment.

Beware the danger of stopping an experiment when it "reaches significance"

When I first joined the team that practiced a form of experiment-driven product development, it was common practice to be constantly watching the results of an experiment as they came in, and run it "until it had reached significance," rather than predetermining, and allowing, a minimum amount of time to pass. This was a huge error on our part.

While techniques *do* exist which allow teams to dynamically stop experiments when a certain level of certainty is reached, they are complex, often misunderstood, and should be avoided when starting out.[2]

Why is it better, then, to not stop an experiment once a level of significance has been reached? The fact is that while you're running an experiment, the randomness of life is occurring, and that randomness is itself, random.

Although you may believe that at a certain point in time, the results you are seeing are "statistically significant" (i.e., that the results fall below your agreed significance level), all this means is that *the evidence gathered so far* happens not to have been affected (enough) by chance. But that could change at any moment. One or two more minutes, and something might occur which throws any notion of "not being affected by chance" out the window.

This line of logic would then suggest that literally as soon as an experiment is declared significant, regardless of the amount of evidence gathered, it should be stopped. But by doing this, you've made a mockery of the significance level and biased the results in favor of your alternative hypothesis. Indeed, the more frequently you check the progress of an experiment, the less evidence will have been gathered, and thus the less accurate the calculation of significance will be.[3]

[2] If you're really interested and highly confident in your abilities, you can read Evan Miller's *Simple Sequential A/B Testing*: www.evanmiller.org/sequential-ab-testing.html, but make sure you understand it before proceeding down that path.

[3] The reason for this is that significance calculations are based on the assumption of a fixed sample size. You can find an excellent explanation of this problem in Evan Miller's *How Not To Run an A/B Test:* www.evanmiller.org/how-not-to-run-an-ab-test.html

Instead, set a statistical power level, a significance level, alongside one other factor, *the minimum detectable effect*, which we'll examine in just a moment, *before* running your experiment. Use these to calculate a **minimum** amount of evidence that must be collected (regardless of whether or not it supports your alternative hypothesis) before any analysis or judgment can be made.

From this "sample size" calculation, you can then estimate how long it is likely to take to gather that amount of evidence. Ensure that everyone commits to a rule of **no peeking** at the results, until such time as enough evidence has been collected. You should **only** stop an experiment once you have gathered that minimum amount of evidence.

How precise do you need to be?

If statistical power and significance level, when taken together, represent the level of certainty you require from your experiment, then a third concept, known as the *minimum detectable effect*, is used to describe how precise a change or difference you will be able to detect when running your experiment.

Determining a minimum detectable effect

In order to work out how long you need to run an experiment for, in addition to the level of certainty you're comfortable with, you need to ask yourself—what size of difference would be interesting, or useful, for us to be able to see? This is what's known in statistics as deciding upon your ***minimum detectable effect (MDE)—***the minimum amount of difference that we could detect within a time period, given an agreed significance level and power.

With belief-led questions, this tends to be fairly simple. Assuming your null hypothesis to be true, when we run an experiment, we wouldn't see any difference between one condition and another. The question is—if we're trying to determine whether there's strong enough evidence to support our *alternative* hypothesis, how much of a difference would be valuable for us to be able to see?

For instance—say we have a click-through rate to articles on a news company's home page of 3%. Being able to detect that a proposed change to the layout of the home page caused the click-through rate to rise to 3.2% (or indeed fall to 2.8%)—is that interesting enough for us? Perhaps not. But knowing that it rose or fell to 4% or 2%—that *might* be valuable, particularly if there's a high amount of traffic in play.

The larger you set the minimum detectable effect, the coarser the change from an existing baseline you'll be able to detect, given an agreed level of certainty. In contrast, if you set a smaller minimum detectable effect, the more precise the level of change from an existing baseline that can be detected, again with an agreed level of certainty.

The upshot of this is that if you set, say, a 10% MDE, then a result which shows evidence of any change less than that cannot be regarded as significant or trustworthy.

Larger MDEs, because they detect bigger movements from a baseline, need less evidence to be gathered. With a larger MDE, you're saying that you're not interested if the needle moves a little—but this will ignore evidence which says your proposed change *would* have a smaller effect.

Smaller MDEs thus need more evidence to be gathered, because you're trying to be more precise—you want to capture smaller differences or changes.

Bear in mind, though, that the smaller your starting baseline, the larger and larger your required evidence will be needed, when setting a smaller MDE. This is because detecting smaller changes on something that is *already* small requires even more precision—even if you're willing to compromise on certainty.

Understanding absolute vs. relative MDEs

When setting a minimum detectable effect, it's important to understand whether you're interested in an *absolute* change or a *relative* change. Some calculators that will help you determine the amount of evidence needed for an experiment allow you to choose between absolute and relative MDEs, but others simply assume one or the other.

From my experience so far, plenty of tools assume you're interested in a relative change, so you need to bear this in mind when inputting your desired MDE. If in doubt, use Evan Miler's sample size calculator, which lets you play around with both kinds, as well as seeing how different levels of statistical power and significance can affect the sample size (i.e., the amount of evidence) needed.[4]

For clarity, let's take a quick look at the definition of "absolute" and "relative" change.

Absolute changes

An **absolute** change is the "real" difference, or gap, between two numbers. It's akin to simply adding or subtracting the change from the number you start with.

So, an absolute change of 1% on 10% would be either 9% or 11%, and an absolute change of 0.1% on 10% would be 9.9% or 10.1%.

[4] You can find the calculator here: www.evanmiller.org/ab-testing/sample-size.html

Relative changes

A **relative** change, however, is the "fractional" difference between two numbers. It's akin to taking a certain slice of the original number and adding or subtracting that from the original number.

For instance, a relative change of 1% on 10% would be 9.9% or 10.1%, because 1% of 10% is 0.1%.

Similarly, a relative change of 10% on 10% would be 9% or 11%.

It can be a little confusing, but the key thing to note is to be aware of which type of change you're selecting. You can set the size of effect you want, regardless of whether it's absolute or relative, but choosing the wrong kind, and mistakenly setting an MDE of completely the wrong order of magnitude, can ruin an experiment's utility.

Selecting an appropriate MDE

Ultimately, then, it's up to you, in the context of your experiment, to set an appropriate MDE. It boils down to how valuable it would be to you, the team, and the organization within which you are working, to know about small or large changes or differences. This, of course, depends on which measure you're interested in and its importance to the health or value of your product.

Calculating scale for belief-led experiments

Now that we have decided upon the necessary certainty for our experiment, using statistical power and significance level, we can combine that with our preferred minimum detectable effect and thus get the *sample size* needed for each condition in our experiment.

Note that this is the sample size for **one** condition—and if we're testing two, as per our null and alternative hypotheses, then we need to **double** the sample size in order to get our total sample size number.

Next, we estimate how quickly we might be able to get to that size. This, in part, depends on the method you're going to choose, and so taking the total sample size into account is going to be a major factor when deciding which method to employ.

Think about how many samples you'd be able to get in an hour, a day, and a week. If you're thinking of running the typical A/B feature test, then depending on the level of traffic to your product or service, you can consider how many hours or days it should take to reach the sample size. If you're considering other methods, such as user interviews, think about how much effort is required to conduct each interview, and thus how many you could reasonably do in a day or week.

All of which is to say—what is the simplest, useful thing you could do, to reach that sample size as quickly and cheaply as possible, without compromising on your agreed level of certainty and precision?

If it seems like it's going to be prohibitively lengthy to reach the minimum amount of evidence needed, then you'll need to compromise on your certainty and precision either by lowering the standards or by seeing what *would* be possible with in a specific amount of time.[5] As mentioned earlier in the chapter, however, this should only be done if there really is no other option.

A note on baseline experiments

The combination of certainty, precision, and MDE work very well in cases where you're hypothesizing a change or difference between an established thing and something different. However, there will be times when you're going to want to understand the potential for something new.

Baseline experiments are ones in which you have a hypothesis—that introducing something new will do something positive for the metrics you care about. The issue at hand, however, is that you're going to introduce it into an environment where that metric currently isn't in play.

For example—the team that I worked with when first practicing XDPD was developing a recommendations product. We had deployed a pop-up window on certain pages across the company's sites, and were seeing a very healthy click-through rate, as well as sign-up numbers for our email service. We wanted to bring the product to a new area of the site, but pop-ups weren't going to be appropriate. So, we designed something that could sit within the page.

Our first instinct was to compare it against the pop-up. But we realized that this wouldn't be a fair test. It wasn't really a case of deciding whether the in-page version was "better" than the pop-up version. That wasn't the question we were trying to answer.

What we were interested in was whether the in-page version would get *any* click-through at all, given it was less eye-catching and further down the page than a pop-up that appeared shortly after a user had started to scroll.

The problem was, when trying to decide our MDE, we realized that the current performance on those pages… was a 0% click-through rate—because it didn't exist. Try calculating a sample size based on a 0% current performance. It's not possible.

[5] The "Power Analysis Calculator" by Rik Higham, at http://experimentationhub.com/hypothesis-kit.html#power-analysis, allows you to set a fixed time window (minimum one week) and tells you what the possible MDE would be as a result.

As a result, it was impossible to accurately calculate a total sample size and thus a minimum amount of time to run the experiment. In which case, we *had* to go for a fixed amount of time—two weeks, in our case—just to get a baseline for how our new version of the product, in a completely new environment, would perform.

It was only after we had the answer to our question—*is this even going to be of interest?*—that we could start iterating and getting back to calculating our scale properly.

Deciding scale for exploratory experiments

So far, when discussing scale, we've focused on experiments which intend to answer belief-led questions. However, a variation of the same problematic outcomes that we saw with those kinds of experiments can occur with exploratory experiments.

Even if you aren't engaged in testing a hypothesis, you would still want to ensure that you set up the experiment to give you the best possible chance of finding interesting knowledge. Similarly, although a single response can tell you a great deal about how someone makes sense of the question at hand, basing a product decision purely on one person's interpretation can be problematic—if it wasn't, we could just ask our most senior team member or stakeholder for their opinion and be done with it.

A "representative" sample

Determining a representative sample size to draw upon for your experiment is still important, but what "representative" means in any particular scenario will depend a lot more on the conditions and criteria you've selected for your experiment.

When determining the necessary scale needed for an exploratory experiment, therefore, you should take into account the availability of resources—existing evidence, likely number of participants, and so on—that would qualify as being relevant for the conditions you've chosen.

For instance, if you were interested in discovering current customers' attitudes to your product, you'd want to ensure a representative sample of customers, so that you get an accurate understanding of the general mood. In order to do this, you'd need to take into account not only the number of customers you have but the number of customers you're likely to be able to get some kind of feedback from, in a reasonable amount of time.

Some may not respond—not because they don't have an opinion but because they're not comfortable or motivated enough to contribute. Others may use it as an opportunity to give you completely different feedback, in addition, or instead of, helping you answer the question you're trying to pose.

Think about the conditions needed, and how much evidence you'd be comfortable basing decisions upon. The basic principles of certainty still apply—no matter the kind of question being posed, the more evidence you can collect, the more certain you can be, but the longer it will take to gather.

Certainty comes in patterns, not anecdotes

What you're looking for, when it comes to exploratory experiments, is *certainty in patterns*. Just as false positives and false negatives can throw you off the scent with belief-led experiments, collecting a number of isolated anecdotes, although interesting, won't necessarily always be helpful in answering the question. Instead, you get certainty from having spotted a pattern—insights that occur often within, and sometimes even across, the conditions of your experiment.

Precision and granularity

Similarly, with precision, it all depends on the granularity of knowledge that you're interested in. Being more precise in this context comes down to how important it is to capture every last *detail* of the evidence. As with belief-led questions, the real trick is not to focus on how *long* it will take to run the experiment but the volume of information required to be able to answer the question effectively.

Would the answer to the question *really* change if you captured the exact time of day, precise location, body temperature, and facial expression of participants? In most cases, probably not.

All of the preceding informationwill help you determine the simplest, useful thing you can do, to help you gather enough evidence in a reasonable amount of time. It may not be quite as neat and tidy as the mathematical formulae underlying hypothesis testing, but it's still an element of your experiment worth designing.

Selecting the most appropriate method

By this stage in your experiment design, you will have filled in all but one of the sections of the experiment card. Now that we know all of the various ingredients for our experiment, there's only one thing left to decide. What's the simplest, useful thing we could do, to answer the question?

The difference between objects of inquiry and methods

When designing your experiment, it's sometimes tough to avoid lapsing into specifying a method up front. For instance, let's say you're proposing that a change in your product would affect a particular metric, or perhaps you want to find out the reasons why users are no longer interacting with a certain feature. Your mind may naturally leap to suggesting an A/B feature test for the former, or a series of user interviews for the latter, even before you've worked out the appropriate hypothesis, measures, conditions, and scale.

Specifying the method up front, however, risks entering into the "method-question-answering" loop I mentioned last chapter. The question you're answering is not whether method X will get you an answer. It's whether the change in the product will affect the metric—or why users feel the way they do.

One way to avoid this trap is to recognize the difference between the **object of inquiry** and the **method**.

The **object of inquiry** is the thing that you're investigating in the experiment, regardless of method. The **method** is the way in which you will perform the investigation.

For instance, in the case of our first example earlier, we're proposing to make an *intervention*—a change that we believe will make a difference. We could do this by running an A/B feature test. Alternatively, we could mock up a paper prototype and test reactions to it. Or, we could gather some user interaction data and then simulate the change. These are our possible methods.

In our second example, we're seeking to perform some *observation*. How might we do this? Run a survey on the site? Go and canvas opinions at a conference? Arrange one-to-one interviews? Again, all possible ways of performing the investigation, but whether they are the most appropriate depends on the conditions, measures, and scale.

Let's try one more example. Think back to the case of Etsy I mentioned in Chapter 5. One premise they identified was that showing more items in search results would be better for business. The *object of inquiry* was their hypothesis—that they would try increasing the number of items loaded on a search result page. The *method* was to run an A/B feature test which would allow them to compare data on how much was bought in the version with more items vs. that with the usual.

Not every experiment is an A/B test; not every A/B test requires a code change

In the world of scientific research, experiments always have hypotheses and always use the framework of the A/B test. In experiment-driven product development, this is **not** the case.

In XDPD, the most important thing is the question, and finding a useful, meaningful answer to it. Regardless of whether we have a knowledge gap we're trying to fill, or we want to test one of our assumptions, claims, or premises, each of these has a question (or indeed many questions!) at their root, and thus they are all candidates for experiments.

The A/B framework is especially important, as we've seen, for our belief-led questions. It can give us important things to consider, mainly when it comes to scale, for our exploratory questions too. That said, don't feel you *have* to force every experiment to be answered via an A/B test.

Similarly, it's important to remember that the "A/B" idea is a framework rather than a method. "A/B" is really just about our two hypotheses—the null and the alternative. Whether we seek to test them via a code change and a new feature, or whether we can collect some evidence without changing the product at all—it doesn't matter. The "A/B" framework doesn't specify the method by which you'll test those hypotheses. So again, even if you *are* designing an experiment around a belief led question, don't feel you *have* to surface that via an actual change in your live product. What really matters is, *is this the simplest, useful thing we could do*?

Simplest, useful thing

In this and the preceding chapter, we've delved deep into how we can try and define what "useful" means, in the context of an experiment. By designing the parameters of our experiment, we know what we would need in order to get an answer we trust, enough that we can base a decision upon it. But *how* should we try and achieve that?

This is where it pays to avoid choosing the method ahead of time. Deciding up front on a method forces you to compromise on the usefulness of the experiment. If you do this, you'll find yourself torturing the experiment design in order to wrangle at least something useful out of it, often at the expense of time, effort, and ultimately, the usefulness of the answer you get out at the end.

Instead, focus on how you might achieve a useful answer. Think of all the *possible* ways you could find the answer, and select the one that is simplest to set up but will still achieve the required conditions, measures, and scale.

Ultimately this comes down to what's practical and possible given your circumstances. Every team is different, with different resources available to them. What might be cheap, quick, and dirty for one can be hideously expensive just to set up, for another.

If it seems that it's not going to be possible to find something which is both simple *and* will be useful, then take a step back. Is the question we've posed too large? Are there smaller, still useful, questions we could ask as a way of taking small steps toward the bigger question?

Similarly, if you choose a method, and then find yourself spending days just setting up the experiment, ask yourself—is this *really* the simplest, useful thing we could do? Is there not something else we could do which would get us a step closer toward the answer?

Again, this may mean that you can't answer the original question you posed in one experiment, but that's OK. Is there a simpler, useful thing you could do, which would still test an assumption, claim, or premise, or help fill in a knowledge gap? Do that. Realizing that a question is too big to answer in one go is still a valuable thing to learn.

Your experiment, your responsibility

We've touched on an element of responsibility previously, when discussing the dangers of collecting user data without any particular reason in mind. Choosing your method for an experiment also has consequences for you and the people who may be included within it while it runs.

The ethics of experimentation are a vast subject and could easily take up their own book. I will, however, touch on one aspect. As much as you should be looking to do the simplest, useful thing in an experiment, you should make sure, too, that you aren't unintentionally causing harm.

Earlier, we discussed the concept of "health metrics"—metrics that aren't always directly under investigation in an experiment but which might indicate serious problems for the long-term health of our *product* if they are negatively affected.

Just as we wish to do no harm to those health metrics, we must consider the ways in which the method we choose might, unintentionally, harm our users. One framework for doing so is to ask the following questions, adapted from the Nuremberg Code[6] by Kim Goodwin:[7]

- *Is it the only way?*
- *Is the risk proportional to the benefit?*
- *What kinds of harm are possible?*
- *How will you minimize harm?*

Asking these questions will help flush out potential issues with the experiment—all part of designing something that will allow you to move, and learn, fast— *without* breaking the things that truly matter.

Running experiments in parallel

It can sometimes be frustrating, waiting for an experiment to finish before starting the next. Depending on your team's capacity, it's possible to run experiments in parallel—at the same time—as long as you avoid breaking the golden rule. Never run two (or more) experiments *on the same thing*, at the same time.

"Same thing" is a little vague, I know. So let's be more specific. Say you've got two different experiments you'd like to run, both involving a user journey across a particular page, screen, or interaction within your product. If you ran both of these at the same time—and crucially, exposed a single user to both experiments within the same session—there's no way of knowing if their response is due to the first or second thing that you're interested in discovering. Assuming this was potentially the case for *anyone* taking part in the experiment, it's now impossible to really know whether their responses are tied to the first or second experiment. Thus, both answers are potentially invalid.

However, if you can guarantee that the two experiments are *totally* independent, or that the chances of crossover are so negligible as to not affect the answers, then by all means, save time and run the experiments in parallel. Just always be aware of the golden rule—experiments **must** be totally independent of each other.

---

[6] The Nuremberg Code is a set of ethical principles to guide experiments involving humans, developed after World War II.

[7] Kim Goodwin (https://twitter.com/kimgoodwin), *Human-Centred Products*, Mind The Product conference, 2018 (www.mindtheproduct.com/2018/11/how-can-we-build-human-centred-products-by-kim-goodwin/)

Summary

Congratulations—you've completed the Design Phase! At last, we're ready to run an experiment, and watch the results come in. We've covered a lot in this chapter, so let's review. What have we learned?

- Deciding on the scale and method of your experiment—**how many** participants, **how long** to run it for, and **how** to gather evidence—deserves some thought.
- Determining the necessary scale has two aspects:
  - **Certainty**
  - **Precision**
- That there are four possible outcomes to an experiment:
  - Evidence for the answer exists, and you find it.
  - Evidence for the answer exists, but you don't find it **(Type II error)**.
  - Evidence for the answer doesn't exist, and you don't find anything suggesting that it does.
  - Evidence for the answer doesn't exist, but you're misled by what you collect during the experiment, so that you believe it *does* exist **(Type I error)**.
- **Statistical power** is the chance of capturing crucial evidence when it does exist, and thus not missing out on something vital:
  - A *higher* power level, the less your chances of missing out
- **Significance level** is the probability of being misled by false positives:
  - A *lower* significance level, the less chance of being mislead.
  - Significance level fluctuates—so don't stop an experiment early.
- **Minimum detectable effect** is the smallest level of detail you are interested in being able to capture.
- All three concepts have an effect on the **minimum sample size** you'll need for your experiment, and thus the time it will take to run the experiment.

- In **exploratory** experiments, we can bear the preceding concepts in mind, but we should aim for
  - A **representative** sample, depending on your conditions
  - An appropriate **level of granularity** for detail, depending on your measures
  - Identifying **patterns**, not simply gathering anecdotes
- When selecting an appropriate **method** for the experiment, you should aim to do the **simplest, useful thing**.
- Remember that you have a responsibility to ensure **no harm** comes to participants in your experiment.
- And finally, that **running parallel experiments is possible**, but they **must** be kept **totally independent** of one another.

Now, we've gone through every part of the experiment card, feel free to start designing some experiments. Don't worry about getting everything spot on the first time around—use all the resources you have to hand, within and outside the team, to help you. The most important thing is to try and to learn—which brings us on to the final phase of experiment-driven product development—the Analysis Phase, which we'll look at in Chapter 8.

PART

3

Aftermath of Experiments

CHAPTER

8

The Analysis Phase

So here we are. You've run an experiment and collected the evidence. Now comes the important part—learning from what you've discovered. If you don't take the time to learn—to interpret the results of an experiment and understand its implications—then everything you've read previous to this has been for nothing.

In this chapter, we'll learn how to analyze and interpret the results of an experiment, how to note the implications of the results, and finally, how to make and communicate the decision for next steps. We can follow a standard analysis process that looks like this:

- Confirm whether or not you have gathered enough evidence
- Determine what the evidence does, and doesn't, tell you
- Consider the implications of what the results mean for your product or service and your process
- Make your conclusions and decisions, and communicate them with context

It can be tempting, at the end of an experiment, to quickly determine a headline result, make a decision, and move on. Depending on the context that you're operating in, and the particular part of your product or service that you're experimenting with, this can be fine.

P. Rissen, *Experiment-Driven Product Development*,
https://doi.org/10.1007/978-1-4842-5528-5_8

But rushing past the analysis stage, concentrating solely on the numbers, and then deciding that this is a "good" or "bad" result can ultimately see you making repeatedly bad calls, based on premature decisions and unrepresentative data. Worse still, without checking things like your health metrics, you might confirm some well-designed hypotheses, but inadvertently do major harm to your product or its audience. So, taking the time not only to determine a result but to set the context around it, and understand its implications, is important.

The Analysis Phase breaks down into two parts. In the first part, we check whether we have enough evidence in order to reach a meaningful answer. If we do, then in the case of a belief-led question, we determine whether or not there is strong enough evidence to reject the null hypothesis in favor of our alternative.

In the second part of the Analysis Phase, we start to interpret the results. By looking at the evidence gathered and, where applicable, the conclusions from our hypothesis test, we can reach a decision not only about what the answer to our question was but what we might want to do as a result.

All interpretations of results are, of course, exactly that—subjective interpretations. And, at the end of the day, you and the team have to make a call on what to do as a result of what you've learned. But by following the steps in this chapter, you can clearly set out what you have, and haven't, been able to learn from each experiment. In the final chapter, we'll run through ways in which you can capture that knowledge in a shareable format, so that everyone understands the background to the decisions that have been made.

Step 1: Do you have enough evidence to reach a meaningful conclusion?

As you might expect, the first thing to do is to check whether you have enough evidence to be able to draw a meaningful, useful conclusion. Based on the principles of scale that we discussed in Chapter 7, you should have an idea of the minimum amount of data needed to be able to test your hypothesis or to ensure you've included a representative set to match your conditions.

With user research or analysis on a static data set, this is pretty simple—keep going until you have enough data. Remember, though, that particularly with the former, it won't always be practical, or even necessary, to gather data from participants at a large scale. You need enough that meets your acceptable levels of confidence—and this really depends on the question you're trying to answer and your own contextual levels of certainty needed.

When it comes to multivariate testing on a live product, however, you're really at the mercy of how your users happen to be behaving at any given point in time. There's almost a mini-hypothesis at work here, because although

you can take an educated guess at how long it will take to gather the minimum amount of data, you won't be sure until it's in progress.

By looking at the average number of users visiting, or interacting with, the relevant part of your product or service, per hour and per day, and comparing that to the minimum data needed, you can take a stab at working out how many hours or days you'd need to run the experiment for. But this, of course, doesn't guarantee that after that time has passed, you'll automatically have gathered enough data. Neither do you want to be checking the data every day, or hour, as this can tempt you to judge the results before they're reliable.

Instead, check in with the data at two points in time. Firstly, a short while after the experiment has begun, in order to confirm that data is actually being recorded. Once that's certain, remember the "no peeking" rule. Do not look at the data until such time as you believe enough data will have been recorded. At *that* point, check the amount of data, but ***not*** the results.

If enough data has been recorded—remember you need enough data in all the variants, if you're running a multivariate test—you can stop the experiment. But if not, have another go at estimating how much longer you'll need to run it for, based on the amount of traffic you've received so far, and check in again at that point.

Step 2: What does the evidence say?

Once we're sure we have enough evidence, we can move on to examining it. First, we'll likely need to **organize** the evidence so that we can understand and interpret it. Only then can we truly test our hypotheses and **answer** the experiment question. Once we've done that, we should check in on our health metrics, to make sure that whatever the answer, we understand the **implications** of what we've done.

Organizing the evidence

Without intervention, it's unlikely that the raw evidence will be in a meaningful state to be able to analyze. Given that we selected our measures and conditions as part of the experiment design, it's unlikely to be a complete mess, though. At the very least, we should have filtered down the evidence being captured, to meet the goals of the experiment. Still, when it comes to analyzing the results, you may have to do some work in order to organize your data in a way that allows us to clearly see the evidence for each condition.

In some cases, this will be fairly simple, though the data may take some wrangling to organize in precisely the way that works best. You can use tools like Google Data Studio, more in-depth analytics packages, or even just a simple spreadsheet, to marshal the data and evidence into a format that allows you to analyze it better.

If you're dealing with more qualitative evidence, you'll still want to sort and categorize evidence. Again, think especially of the conditions from the experiment—these will likely be your categories. Measures, when it comes to qualitative evidence, are sometimes harder to clearly identify within the evidence (though not impossible). This is, of course, where putting some thought into designing the experiment—constructing a decent hypothesis, specifying the conditions and measures—*before* running the experiment, can help, as it gives you a head start in tagging, categorizing, and sorting the evidence in a way that will be useful later on.

Answering the question

With the evidence and data in a format that can be easily interpreted, now is the time to look back to the question, and if applicable, the hypotheses, and determine the answer. Can you reject the null hypothesis, in favor of your alternative? What is the most plausible answer, based on the evidence available?

Using a p value to check the strength of the evidence

In the previous chapter, when discussing scale, we looked at different possible outcomes for an experiment. One of these was that evidence for an answer could be collected during an experiment, but when subjected to scrutiny, it turns out that this evidence was purely down to chance rather than anything meaningful or significant.

We discussed the use of a *significance level* to determine how certain we wanted to be, if it looked like there was evidence to support our alternative hypothesis. The lower the significance level, the more certain we would be—but the more evidence we would need to collect, in order to rule out chance.

If the significance level is our "acceptable" probability of false positives, regardless of what actually happens, then the **p value** tells you what the *actual* probability is.

That is to say, the p value is calculated from the evidence gathered, together with the scale of said evidence, and working out how likely it is that this *particular* set of results occurred due to chance.[1] What we then need to do is compare the **p value** to our **significance level**.

As mentioned in the previous chapter, it's up to you to set your acceptable level of significance, but a commonly used one is 5%, or 0.05. This means that there's only a 5% chance that the results could have occurred due to chance. So, if the p value is **less than** 0.05, then we can say that the chances of what actually was detected during an experiment is under 5%, so we're happy to thus reject our null hypothesis and accept the alternative.

If the p value is **equal or greater than** the significance level we set, the safest thing to do is to stick with our null hypothesis.

Seeking the most plausible answer

Comparing the p value to the significance level is one way of determining the **plausibility** of an answer. When it comes to answering the question, regardless of it involving a hypothesis or not, you should be looking for plausibility—the strongest, most likely answer and explanation.

There's still a chance, of course, that you're wrong in your conclusion. You can never be 100% certain—and future circumstances may change what you've learned—but answering a question, backed by plausible, strong, and ideally statistically significant results, should be the standard.

Remember, too, that being wrong about a previously held belief, or finding that a knowledge gap just can't be filled, even with a well-designed experiment, is not failure. You've still learned something that you can take forward into your future work.

Checking the health metrics

Experiments are a relatively low-risk way of testing out whether your existing beliefs, and your current plans for the future, are leading you down the path to success or failure. They can never guarantee success, of course, but by testing your ideas out in relatively safe environments, you can throw away the obviously bad ideas before they lead you to ruin.

[1] The actual calculation of a p value is out of scope for this book, but there are plenty of online resources which can help, for instance this, by Rik Higham:
www.experimentationhub.com/p-value.html

Bad ideas, though, aren't always immediately obvious. Hindsight is a wonderful thing—short-term gains can mask an underlying fatal blow to the health of your product or service. Without considering whether an idea will have unintended consequences, outside of the immediate remit of your key question and hypothesis, you can end up shooting your success in the foot.

Once the experiment has run its course, then, as well as checking the results to see whether your main hypothesis has been proven or disproven, you should couch the result in the context not only of the amount of uncertainty surrounding the result but also the impact on your health metrics.

Although we talk about these in terms of "metrics," they can be equally applicable in more qualitative arenas. If you're seeing significantly negative reactions in and around your user research sessions, then regardless of the answer to the main question you're investigating, you really should consider whether pursuing this avenue will damage your team, the organization, or the brand, in the long run.

Similarly, you should be considering the effect on the users themselves and society. It's often difficult to determine these wider impacts in the time frame of an experiment—certainly it's rare that you'd uncover something significant—but early warning signs can be spotted and taken into account for the future.

Saving for later

Evidence from experiments of all kinds can be valuable in the long run, so it's important to capture and save the evidence, tied to the details of the experiment (the hypothesis, the dates it was run, and the various measures of precision and certainty). The experiment card format, when captured digitally, can be used as a datastore of links back to the raw data in this way.

Evidence from user research can be especially valuable in this regard. In the immediate aftermath of an experiment, you'll be most focused on organizing the evidence so that you can test a particular hypothesis and answer the question. However, it's likely that there will be other nuggets of interesting information within the interviews or research that can inspire new questions and, in turn, new hypotheses, knowledge gaps, and experiments. Therefore, as you're sorting through the evidence from one experiment, don't forget to identify other points of interest that you might want to come back to.

Step 3: Consider the implications

By this point, you'll know whether your hypothesis, within the boundaries of certainty provided by the result, was correct or not. It's important to state this clearly up front. This is the result, with this level of uncertainty and in these particular circumstances. The next step, then, is to consider what this result implies.

To do this, you need to take a step back from the hypothesis and consider the question that underpins the experiment in the first place. You're now able to answer that question, albeit with some small print that sets the context. That's our first implication—that we have an answer to the question, and that this gives us *new knowledge* to build upon.

The secondary implications are perhaps more important. The answer will have knock-on effects, which essentially boils down to another question. *How does the answer change what we thought we knew?*

Perhaps we've confirmed and bolstered our belief in something we already thought. That gives us more confidence in a direction of travel and can thus imply we should double down on this line of research rather than switching to a different track.

Alternatively, although the answer may confirm a belief we already had, or fill a gap in our knowledge, it may suggest that the belief needs to be tempered by the fact that the effect—the change or difference—is not as large as we believed it to be, or that the knowledge gap wasn't as important to fill as we had thought. It's possible that we can have a belief confirmed, only to learn that although we might be "right," perhaps our time would be better spent, now, looking at something that may have even more benefit to the future of the product, service, or our users and society.

In contrast, the answer to the question may completely contradict something we thought we knew. Although this can be painful, it's still extremely useful. It causes us to wake up to the fact that we need to consider changing our direction. At the very least, it should raise a whole new set of questions which examine our other beliefs, and whether they, too, should be reconsidered.

In this way, you can see that the results of an experiment, the decision to reject or stick with a null hypothesis, and ultimately the answer to a question can have a major impact on product strategy.

This is a fundamental lesson of the experiment-driven approach. People often talk of the idea of "pivoting" in terms of a major change of direction—from game company to photo sharing application, or from DVD rental service to video streaming behemoth.

Although these kinds of major pivots do happen, the truth is that whatever plans and road maps you make, you need to be prepared to perform potentially hundreds of tiny pivots during the lifetime of your work, as new knowledge, and new questions, arise. Sometimes, you won't know what you're going to do, until you reach the point where you have enough knowledge to make a decision. And that's OK.

What does the experiment not tell us?

New knowledge is wonderful, but it's also important not to read too much into an experiment. We've talked about acceptable levels of uncertainty and the undeniable nature of results being valid within a certain context. But it's also important to clearly communicate what the result does *not* tell you.

In some ways, this is a case of being ahead of the game. It's impossible to completely prevent misinterpretation of experiment results. Both stakeholders and team members, even you, will leap to conclusions which far outstrip the actual evidence from an experiment, from time to time. The more you can do to state up front, and share, what we *cannot* prove from an experiment's results, the more you can lessen the impact of false conclusions.

It can seem disheartening, going through the things you can't prove from an experiment. It can lead you to question the value of having done the experiment in the first place. But this is a blessing in disguise. As noted, it stops you from making false assumptions which can ruin your next steps. Secondly, it, too, is fuel for the fire for future experiments. Working out what this experiment was unable to tell you presents the obvious next question—how might we be able to learn this thing instead? And how important or influential would it be, if we could learn that?

So, presenting a list of things you can't tell, from the results of an experiment, not only sets sensible boundaries on the experiment's influence but also generates a whole slew of possible new avenues to consider.

What did we get wrong, or what could we do better?

As with any process, experiment-driven product development is rarely done perfectly right, the first time. Even once you've got into the rhythm of designing, running, and analyzing experiments, you're bound to make mistakes along the way. You should always take the time, therefore, to consider what you'd do differently next time.

Maybe it's a part of the process that didn't quite work. Maybe it's a step along the way that you forgot to do or were confused by. Maybe not all of the members of the team were involved, or fully bought in. There's always room for improvement.

There's also likely to have been missed opportunities—things that you only realized during the running of an experiment. You didn't necessarily get anything wrong, but while running the experiment, you might realize there was something you hadn't considered, or indeed a completely new angle on an existing question or issue that arises. Sometimes it's a matter of process improvement.

All of this is the familiar ground for team retrospectives, which tend to occur on a regular cadence, perhaps at the end of sprints, or more loosely at regular intervals in time. These ceremonies tend to focus on improving general team processes, looking back over the achievements of a team over time, or resolving the health and happiness of a team.

In contrast, you should consider doing "mini-retrospectives" at the end of every experiment. These can be as lightweight as quick conversations with the team (either as a group or in one-to-one settings), or perhaps in the guise of a small survey that you can pass around. These will help you spot parts of the experiment process that can be improved, opportunities that were missed, or questions and concerns that team members may have about the whole approach.

What else is intriguing?

Finally when it comes to analysis, it's useful to consider what else might be hidden in and among the results. The main focus is, of course, on answering a question, and on testing the hypotheses. But as well as considering what you have and haven't been able to learn, you should also take note of any interesting or intriguing patterns and questions that may emerge from the evidence you've gathered.

These are likely to be things that either you've not considered at all before or might be useful indicators toward an answer for another question already on your backlog. Either way, although it's unlikely that the incidental evidence or data that you've gathered from a particular experiment can prove something else definitively, capturing that evidence so that you can come back to it later is worthwhile. Similarly, if it is something that is completely new, being able to come back to this, and feed it into the beginnings of the whole experiment-driven product development process, is key, if it is ever to be useful to you.

Step 4: Reaching conclusions and making decisions

Eventually, you're going to need to draw the analysis stage to a close. Although there will always be more to discover, more to refine, and an ever-present level of uncertainty, getting stuck in the analysis stage can be just as fatal as rushing down the wrong path. The trick is to analyze quickly but carefully, reach sensible conclusions, and make a firm decision.

The conclusion, or headline result, should be stated clearly and simply. From the experiment, under certain conditions and circumstances, the alternative hypothesis will have either been adopted or rejected. Alongside the result,

you should state the significance level that you decided upon and the p value. With these two data points beside each other, you can demonstrate the solidity of the result. The lower the p value, the more solid the result.

This doesn't mean that the result should never be questioned, but, assuming that you designed the experiment so that the result could be useful in more than the exact circumstances under which the experiment was run, it can be safely assumed to be true for the foreseeable future.

Regardless of all this, the final step is to make a decision. Given the result, given the knowledge, and given the degree of uncertainty, what are you going to do? This could mean you start or stop doing something. It could mean that you change your plans for what you were going to investigate next. It could mean that a potential course of action for another team, or other stakeholders, needs to change. Whatever it is, making a decision is something that has to happen. Otherwise, you're gathering knowledge and never acting upon it.

Summary

In this chapter, we've discussed what to do once an experiment has reached its conclusion. The process of analyzing and deriving insights from an experiment is the main way in which the experiment-driven approach yields value. Without analysis, you're performing to the gallery, running experiments, and never learning.

We ran through the various steps in the analytical process:

- Confirming that you have enough data to reach a meaningful, useful answer
- Examining the evidence in order to state what you can, and can't, learn from it
- Thinking through the impact, and implications, of the answer you've discovered
- Reaching a conclusion and making a decision on how this affects your future plans

Our experiment is now over. We've been through the Planning, Design, and Analysis Phases—the core parts of the XDPD approach. In the final chapter, we'll consider the implications of the result a little more—how these affect your backlog and priorities. We'll also look at a useful format for sharing the results of your work with stakeholders and other interested parties. With the implications for your backlog and priorities realized, and the results communicated, this will close our examination of XDPD.

CHAPTER

9

Where Do We Go from Here?

Your experiment is complete. You've taken the time to analyze the evidence and determine whether your hypothesis has passed the test, or otherwise. You've noted what you've learned, what you haven't been able to determine, and any intriguing questions that have arisen. Most importantly, you've reached a conclusion, and made a decision.

So, now what? Move on to the next item on your backlog, run experiments, and rinse and repeat until the end of time. Job done, right? Book over, thank you and goodnight, yes? No. Not quite.

The decision you've made has consequences and knock-on effects. Namely, it's important to clearly communicate the decision you've made, and to take a fresh look at your backlog, after each experiment, and see how the decision could affect your priorities, both immediately and in the mid to long term.

In this chapter, therefore, we'll discuss

- How to reexamine your backlog, based on decisions after an experiment
- How to communicate your work involving experiments to your stakeholders

P. Rissen, *Experiment-Driven Product Development*,
https://doi.org/10.1007/978-1-4842-5528-5_9

Reexamining your backlog

Experiments are a structured way of asking questions. Why are we asking questions in the first place? To gain knowledge, to test our assumptions, to understand the world around us, and ultimately, to help guide our decisions. Those decisions affect what we do after an experiment has ended.

Learning for the sake of learning might be pleasant and can help inform a business' wider strategy, but most of the time, you'll want to be applying that new knowledge to affect what you do next. And if what you've learned hasn't changed what you do in the future—or at least change your *confidence* in what you were planning to do next—then the value of that knowledge is questionable.

Thus, after an experiment has been run, it's worth taking another look at your planned backlog, and considering whether those plans need to change. In a moment, we'll run through a process for evaluating the items on your backlog in light of what you've learned. Before doing so, however, it's worth saying that this reconsideration doesn't strictly need to happen after every single experiment, nor does it always have to be a formal exercise each time.

As ever, you should be pragmatic and use your best judgment. The more often you reflect on your backlog, given what you've learned, the better, but sometimes a single experiment in isolation won't be enough to warrant such reflection. Partly, this depends on the length of time it takes to run several experiments.[1] If you can get through a small number in a reasonable amount of time, then you can take some time to reflect after a series of experiments.

You'll have to weigh up the risk that you may waste some effort on experiments, if you've not had the chance to contemplate their consequences, but one approach might be to regard this reflection session in the same way as a team retro or backlog "grooming" ceremony—schedule it in as a regular thing, at least to start with, so you can get used to reflecting, then adapt to suit your team's rhythms.

A process for reflecting on your backlog

In Chapter 5 of this book, we examined how turning claims, assumptions, premises, and knowledge gaps into questions can help you build a backlog to explore, which in turn guides your product development. Two aspects of this process can help us when it comes to reflecting on your backlog in light of the results, conclusions, and decisions from an experiment.

[1] Running a few experiments in parallel is an option, of course, but you should always be mindful that running more than one experiment at the same time can affect the results. So, think carefully and try to choose experiments that won't clash.

Step 1: Re-score and re-prioritize

The first is the way in which we prioritized among a pile of questions in the first place. Recall that we had specific criteria, ironically in the form of questions themselves, which we used to score the importance of answering each of the questions on our backlog. The criteria were

- **Confidence**: How certain is the team that we already know the correct answer to this question?
- **Risk**: What's the danger if we're wrong about the premise, or if we don't answer the question soon?
- **Impact**: If we know the answer, and/or we're right about the premise, how much benefit might that bring to the team or to our users?
- **Interest**: What appetite is there within the team for finding an answer to this question?

The idea when we were first prioritizing the questions on our backlog was to give each item a number of points against each of the preceding four questions. Although adding together each of these values gives you a total score which helps guide your initial prioritization, each of the individual scores for an item should also be recorded, not only for posterity but for the precise purpose of this moment of reflection.

The key thing you'll want to do, therefore, when reflecting, is to examine whether these scores should be updated. Given your new knowledge, or as a result of the decision that has been made off the back of an experiment, you need to ask whether each score should be reduced or increased. Ultimately, you need to reask yourself the four questions:

- Is our confidence that we already know the answer to this question *reduced or improved?*
- Has the risk of our supposed answer being wrong *reduced or improved?*
- Has the potential impact to the team, our stakeholders, or our users been *reduced or improved?*
- Has the appetite of the team for finding an answer this question been *reduced or improved?*

Re-score the items, sum them up to give each item a total, and then re-rank the items on your backlog as per the new scoring.

Step 2: Reshaping your backlog

The other aspect of our original prioritization session which is worth revisiting when reflecting is the concept of the "broad" vs. "deep" backlog. Remember that although, in some cases, a single column of ordered items in your backlog can help provide focus and clarity as to your road map and priorities, a truer reflection of the process of knowledge accumulation is to reflect your backlog in a treelike structure of nodes and branches.

Previously, we defined a "broad" backlog as one reflecting the team's desire to explore several different (often shallow) chains of connected, but distinct questions. In a broad backlog, you might start with several high-level questions and seek to either explore some of them in parallel or switch tracks in a more serial fashion, after each experiment.

If you're going to do this, then updating your backlog tree is vitally important. You need to clearly reflect the updated priorities on the backlog, as you're likely to be switching contexts often. Although each "branch" of the backlog may seem unrelated to each other, it's also likely that connections, or even reorganization of the branches, will emerge over time as you gain knowledge.

It's also very likely that, given results and decisions from each experiment, you'll want to "prune" branches of the backlog tree. Given a particular experiment, you may decide that some of the original premises or questions you had are no longer applicable. Technically, they're probably things that you should keep in cold storage somewhere, so that they can be revisited in future, but if you've made a particular decision that you're *not* going to do something, then you should remove anything related to that, from your backlog tree—otherwise it's simply a confusing distraction.

A "deep" backlog is one where you have perhaps one, or maximum two, high-level questions, but a much larger number of highly related questions which may even be chained together in a sequence of causation, not just prioritization. Just as with the "broad" backlog tree, after re-scoring your items, you should consider whether the new scores or the decision that you've made means that some of those items are now no longer relevant. If so, drop them.

Similarly, it could be the case that although the question is still relevant, the sequence that you had them in should be changed. This could be because the supposed chain of causation and connection is actually different, given your new knowledge, or it could be because the urgency and appetite to answer one particular question have boosted it or demoted it to a new position in the list.

Step 3: Consider new raw material

Not only is it important to reflect on how the results of an experiment, or a decision, affects your planned backlog, but it's also useful to incorporate new premises, assumptions, claims, and questions.

As mentioned in the previous chapter, it's likely that given your new knowledge, there will be new, intriguing possibilities for investigations and new assumptions or premises that you're likely to make once you've seen the results of an experiment. These new items need to be prioritized in the same way as we've described here and in Chapter 5, but they themselves might also have knock-on effects upon your existing items.

Existing questions may no longer make sense to explore alongside new ones. In which case, decide which track you're going to take (and make note of the road *not* taken, so you can revisit it, if needs be, in future). New questions could result in completely new branches of your backlog tree. They could turn a deep backlog into a broad one. And of course, these new items might receive higher scores than existing items, so you'll want to rejig the positions of existing items to fit these new ones in.

When it comes to considering those new items, you shouldn't forget, either, about the principle of *simplest, useful thing*. Normally we think about this mainly once an experiment has been chosen to be worked on, and it influences the method by which we'll be seeking to answer the question. But it's also worth keeping in the back of your mind when re-prioritizing.

This is because it's often easy to get carried away with adding many new items as a result of an experiment's results, and can also feel overwhelming as you run the risk of questioning everything that went before. The feeling of needing to scrap everything and start again can be tempting but also often foolish.

Although you won't want to nail down exactly how you plan to go about answering each of the new items, you should view them through the lens of simplest, useful thing. If you can't, in a relatively short space of time, think of potential ways that you could explore the question in a simple, but useful way, then perhaps it needs to be reworked and broken down into smaller elements.

Similarly, if there is no simple, useful way in which you can answer the question, you should question what the real value and risk would be in choosing to explore or ignore this question. If it's still considered valuable or high risk, then this is something that should be communicated outside of the team. Essentially, you're saying that you can see value in answering the question, but with the resources available to you, you don't believe you can answer it effectively.

Sometimes, too, it could be such a profound, complex, or high-level question that a single team couldn't possibly answer it. Here, you should really think whether there are smaller parts of the puzzle that you can gain some useful

knowledge from. Knowledge which won't answer the big question outright but will perhaps at least contribute to progress. Perhaps it needs several teams, across the organization, to bring their own skills to bear on answering it. Here lies the realm of cross-team and even cross-department, collaboration, coordination, and strategy.

Sharing and planning your work

Regardless of whichever process, methodology, or framework you use to organize your team's work, sharing your progress and future plans is important. How so?

Firstly, it helps the team achieve clarity, both in terms of looking back at what they've done and setting the objectives for work in the future. Secondly, it also helps reassure your stakeholders that you're doing useful work and communicates to them the value of what you've learned already.

Sharing your work and future plans is best done in a regular pattern, so that both the team and your stakeholders get used to communicating regularly with each other and so that everyone is aware that commitments are made and focuses and objectives are set.

Plans can be made but can also be changed—being as clear as possible on what you're planning to work on, how close you stuck to that plan, and, crucially, why you deviated helps build trust and understanding on all sides.

Sharing your work also goes hand in hand, to some extent, with planning your future work. Whether you see this as analogous to "sprint planning" or not, setting objectives within the team for what you want to achieve in time for the next shareback or demo[2] helps set expectations and rallies the team around a mutual set of goals.

In some ways, this pattern of working can be seen as almost a "meta" experiment, with shareback sessions being hypothesis-driven and with experiment results being shared. In this section, let's look deeper into how this is the case.

Setting targets

If you've used the sprint concept within agile ways of working before, you'll be used to the concept of sprint planning. This is a session in which you work with the team to select items from a backlog, so that you can bite off a reasonable chunk of work within a defined period of time—all of which should get you closer to achieving larger objectives.

[2] Lots of different words are used for the same concept—shareback, showcase, demo.

If you use a more "Kanban" style, whereby work continuously gets picked up from the backlog, based on what can sometimes feel like an invisible, behind the scenes process of backlog prioritization, it's easy to miss the value of setting targets and sharing progress.

Of course, targets can shift, and not meeting those targets can be legitimate. At the end of the day, though, it's important to communicate why certain targets have been chosen, and likewise, *why* those targets weren't met, when it happens.

This is where treating your working process itself as a series of experiments can be useful. Your targets are essentially a series of hypotheses that you're making. By the time of the next shareback session, we believe we can achieve these things. Just like an experiment, you should aim to communicate why you've chosen to work on these things.

And just like an experiment, when you put together the messaging for the shareback session itself, you'll want to share not only the results—whether you achieved those targets, and how you did so—but an analysis of the meaning of those results, that is, whether your hypotheses were wrong, why, and how these results are going to influence your future plans.

When should you set your targets?

Either immediately after or, preferably, in the run-up to your next shareback session.

It might seem like the obvious time to do this would be after a shareback session, be that immediately after or the following day. However, as we've mentioned, sharing your future plans—your hypotheses about what you want to achieve by the time of the next shareback session[3]—is an important part of what you communicate.

If you only agree as a team on your targets for the next session, after the previous one, you've missed a useful opportunity to communicate those targets. And by not doing so, you run the risk of making it seem like you're deliberately trying to keep your work covert.

Not sharing those targets ahead of time suggests you've made up the targets to cover off everything you achieved and thus appear like you've never failed. Which might seem appealing at first, but this in turn leads to your stakeholders

[3] For clarity, it should be noted that these hypotheses aren't necessarily the same as the ones that you're trying to test within your work—these ones are hypotheses describing your beliefs about what you can achieve as a team rather than your prior beliefs about aspects of your product.

being unaware of the difficulties you may be facing, as a team, in achieving your goals. Yes, they may judge you on that—but they might also be able to help clear the route.

So, in the run-up to your next shareback session, gather the team together, and work out what you want to aim to have achieved by the time of the following session. Once you've done that, a clear and simple format for your shareback session can emerge. A regular format, too, helps set expectations and smooth communications both within the team and with stakeholders.

A structure for regular communication

Certain principles can help structure your shareback sessions. Whether you create this structure as a series of slides for presentation, or as a document structure for regular emails, the principles should be the same. So, let's walk through them.

A reminder of why we're here

It's always a good idea to start with a reminder of the team's wider or longer term goals. Not only their mission statement, purpose, or vision, if you have one, but also, for instance, your goals for the quarter.

Showing this up front sets the scene and establishes important context for the work you've done until this point. It also helps people understand your priorities and the decisions to concentrate on, or ignore, certain hypotheses, questions, or indeed other work, within a set time period. It's equally useful when sitting down with the team to prioritize targets for the next shareback session.

Targets: What you wanted to achieve, and why

As already discussed, setting the targets ahead of time allows you to share them early. Again, this is important for expectation setting. Describing *why* you've chosen to focus on those targets should also reflect the value you feel you can bring from getting these particular things done.

This can help reassure stakeholders that the targets the team has chosen, or the work that's been prioritized, will add value, even if at first it seems less obvious.[4] Think of these as your hypotheses—by working on these, or by achieving this, we'll deliver this change or learn more about this difference.

[4] Fixing back-end systems, improving accessibility, or clearing up tech debt are good examples of work that stakeholders sometimes question the value of but can dramatically improve the success of your product.

Ideally, you should be able to clearly articulate how your targets move you closer to achieving your long-term objectives. This is also often a good time to have a brief question and answer discussion, in case anyone has strong feelings about whether anything else should be in there, and what might have to be moved out, in order to incorporate that.

Progress: What you actually achieved, and why

Now is the time to compare what you planned to do (i.e., your targets that were revealed in the previous session), against what you actually did. These are the results of your meta-experiment—were you able to achieve your targets, and if not, why not.

Again, explaining why, especially in cases where what you did differed from your plans, is important for building trust and clarity. In cases like this, it also helps to explain not only your thinking for why you diverted from the plan, but, just like with an experiment, by analyzing the "results," you can learn lessons about whether those targets were reasonable or otherwise.

Don't forget here, too, to actually demonstrate what you've done. The simplest, useful thing that you've done, no matter the method, to learn. Screenshots, photos, audio clips, and videos—anything which gives a flavor of the format of any experiments you've run during the period. You can also show the completed experiment cards, together with their headline results and any decisions made as a consequence.

Other work: Collaboration and distractions

It's often the case that there will be some amount of work done by the team, during the period between shareback sessions, that doesn't necessarily work toward the targets set.

Sometimes, this effort can help *other* teams achieve their goals, and sometimes, this work doesn't appear to provide any value at all—it's a distraction and an annoyance. In either case, it's important to surface that work within a shareback session. If it's a case of collaborating to help another team achieve its' goals, that's something worth shouting about.

Equally, if the work has been an unhelpful distraction, as mentioned before, surfacing it is still useful. By doing so, you can ask for help in unblocking issues or problems. Stakeholders themselves may offer to help remove these obstacles. All of which builds trust and empathy between the team and stakeholders. Of course, if you find constantly spending the majority of your time on nonproductive work, then there's a wider discussion to be had about how this can be reduced and improved.

What's next: Continuing work, about to be kicked off, on the horizon

The final section of the shareback should look ahead to the future. Here's where you reveal your targets, and reasoning, for what you want to achieve by the next shareback session.

More explicitly, it's worth calling out work that is already in-flight and will be continuing. Not only does this cover work which is still in progress, but it can be a useful time to share progress on planning new experiments. You can use the experiment card format to share your current thinking on soon-to-be-run experiments—the question, the why, and any progress you've made in determining the hypothesis, measures, conditions, and method.

Given that you've already covered off "finished" experiments, you can remind your stakeholders about any experiments that are about to start, or are likely to begin, during the time before the next shareback session. Again, this reinforces an accurate impression of the work that will be taking place (i.e., that you've already committed to), but it also gives prior warning, especially in cases where the stakeholders or people they know may experience being purposely or inadvertently included within the population of the experiment itself. "*Be prepared—we're experimenting*" is the message here.

Finally, just as you started with the long-term goals, it's worth closing on what work is on the horizon. This work will include things that you've not yet committed to, but it's your current thinking about what might happen *after* you've met the targets you're setting this time around. It's also useful here to flag any upcoming collaborations or distractions that you know about ahead of time.

Conclusion

In this and the preceding chapter, we've covered the "aftermath" of experiments. The need to analyze the results of an experiment, capture any thoughts arising from those results, and make a decision about how to proceed, we covered in Chapter 8. This time, we covered the wider ripples caused by those decisions.

We discussed how your backlog should be updated to reflect what you've learned and what decisions have been made. You can do this as often as you'd like, and make it as formal or informal as you want, but the important thing is to use some kind of process which takes into account the following factors:

- Whether what you've learned, or your decisions, reduce or improve confidence in the other premises on your backlog
- Whether the results reduce or mitigate the risk from *not* prioritizing certain experiments

- Whether the results or decisions made affect the appetite to explore particular questions on your backlog
- Whether the results or decisions indicate a change in the value that could be gained from answering a particular question on the backlog

You can use a points-based system, and/or have a discussion with your team, but you should be open to your backlog, and your priorities, changing as a result of the experiments you run. This can include, don't forget, new questions and premises appearing, as well as certain existing items being no longer applicable.

Secondly, we covered how to communicate the work you're doing in a way that provides clear direction, reasoning, trust, and empathy between the team and their stakeholders. The way you plan your work can be regarded as an experiment itself:

- You set targets, ideally prior to a shareback session, which are your hypotheses for what you can achieve. The work you do between now and the next session is the process of testing those hypotheses.
- You report back on the results—whether those hypotheses turned out to be true or false. Within this, you can include the results of the experiments that were undertaken during this time.
- You analyze those results and share any lessons learned—what *else* did you end up working on?
- And finally, you share your new hypotheses for what you want to test in the upcoming days, weeks, months, and so on. Here, you can also give prior warning of any experiments that will kick off in the near future.

By regularly reworking your backlog, you can make an effort to reduce wasted effort and keep the team on target, while still incorporating new knowledge. By communicating your plans, your intentions, and what actually happens, you extend the value of that knowledge beyond the immediate team and influence not only your own successes but those of the entire organization.

It's simple and useful. These are your very first simplest, useful things—the guiding principles that allow you to plan and implement the experiment that shelters everything else. The experiment of trying experiment-driven product development.

I

Index

P. Rissen, *Experiment-Driven Product Development*,
https://doi.org/10.1007/978-1-4842-5528-5

F

G

H, I, J, K

L

M

N

O

P

Q

R

S, T, U, V, W, X, Y, Z