Microsoft

Scenario-Focused Engineering

A toolbox for innovation and customer-centricity

Best practices

Austina De Bonte and Drew Fletcher

Praise for *Scenario-Focused Engineering*

"Breakthroughs often result when diverse disciplines collaborate to solve old problems using new perspectives. Scenario-Focused Engineering is just such a breakthrough. I'll never see software design the same."

—Eric Brechner, Development Manager, Microsoft Xbox, author of I.M. Wright's Hard Code

"If your team focus is dominated by what you want to make, without enough consideration of why or for whom, this book is for you. Revitalize your team and your product by using these rigorous techniques that put the customer at the center."

—Chris Pratley, Director of Program Management, Microsoft Office, creator of OneNote and Sway

"De Bonte and Fletcher have astute insight into how engineering teams build products today. They expertly lay out a compelling approach to the creation of desirable products and services through the embrace of Scenario-Focused Engineering. Read it and you'll want to start using Scenario-Focused Engineering in your development processes immediately."

—Gayna Williams, founder, Swash Consulting

"If you are new to customer-centricity or an expert with decades of experience, this book is a great addition to your library. It demonstrates that customer-centricity is the responsibility of everybody in the organization and gives readers strategies and ideas to make this happen."

—Charles Bennett, CEO, NextTen

"Microsoft has gradually shifted from a feature-focused approach to product engineering to a more user-centric, scenario-focused approach. The shift was both profound and difficult. The SFE training they drove was instrumental to Microsoft meeting these challenges. In this book you'll get the distilled lessons of that enormous undertaking."

—Charles Eliot, Head of Engineering Services, Skype

"In this impeccably organized book, Fletcher and De Bonte combine practical wisdom and highly refined techniques to produce a hands-on guide that will enrich the design room as well as the classroom. A smart, easy read."

—William Storage, Visiting Scholar, Center for Science, Technology, and Society, UC Berkeley

"One of the toughest challenges designers face is promoting the value behind building end-to-end scenarios rather than hundreds of glitzy, yet disconnected features. SFE throws the old engineering processes out the window and replaces them with a common language, tools, and techniques that enable development, test, program management, marketing, and design to work together to deliver a cohesive, end-to-end experience. The program transformed our organization from the top down."

—Bethany Kessen Doan, Principal User Experience Design Consultant

"The concepts of SFE are presented in a digestible and easily adopted way for people of all levels of experience. Having seen firsthand the impact the concepts can have on an engineering team, I am a big supporter of this way of thinking."

—Sheri Panabaker, Principal User Research Manager, Microsoft Surface

"Three years ago, we decided to pilot the use of Scenario-Focused Engineering for our division. The improvements were almost immediate, and customers love the outcome. There's no going back."

—Jeff Comstock, General Manager, Microsoft Dynamics R&D

"With great examples and a proven approach, this book provides a great roadmap for really learning about your customers and how to build great products for them."

—Mike Kelly, Managing Partner, Tech DNA

"Teams will greatly benefit from this book by shifting the primary focus from feature lists to 'who would use this product' (target customer) and 'why/how' (scenarios)."

—Raja Abburi, CEO, Navaraga Corporation

"I saw firsthand how engineers became more deeply involved with customers. SFE was a great step forward."

—Mike Tholfsen, Principal Engineering Manager, Microsoft Project

"Chockfull of common sense, SFE was controversial because taken as a whole it pushed for a much-needed culture shift at the company. We used to dream up scenarios to match the features we wanted to build. SFE helped teach the company how to start first with real customer needs and then design for the right scenarios."

—Kent Lowry, Principal Design Research Manager, Microsoft Office

"There are a handful of moments in life that make an indelible impression on one's memory. One of those moments for me was when Austina wrote at the top of the whiteboard, 'To help design a product customers crave.' SFE helped Microsoft transform, and it can help you as well."

—Michael Corning, Senior Data Scientist, Microsoft Skype

"This book will be a priceless asset in helping me apply SFE at my company."

—Arne de Booij, UX Strategist, SDL

"For those of you trying out SFE for the first time, trust that this system works. Trust that getting your engineers involved in the design process helps them understand the problem they are trying to solve. Trust that iterating and getting feedback on these iterations from customers can be done quickly. Trust that you will have that 'aha moment' when you show a design to customers and they are ecstatic. It's then that you realize that SFE works."

—Kevin Honeyman, User Experience Lead, Microsoft Dynamics

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Control Number: 2014946865
ISBN: 978-0-7356-7933-7

Printed and bound in the United States of America.

First Printing

Microsoft Press books are available through booksellers and distributors worldwide. If you need support related to this book, email Microsoft Press Book Support at mspinput@microsoft.com. Please tell us what you think of this book at http://www.microsoft.com/learning/booksurvey.

Acquisitions Editor: Devon Musgrave
Developmental Editor: Devon Musgrave
Project Editor: Devon Musgrave
Editorial Production: Rob Nance and John Pierce
Copyeditor: John Pierce
Indexer: Lucie Haskins
Cover: Twist Creative • Seattle

Contents at a glance

This page intentionally left blank

Contents

Foreword

As a leader of a large organization, I know that change is hard. Systems are built up over many years to prescribe the what and how of things. Organizational structure, design artifacts, and processes are all put in place with good intentions. Each individual decision is often the right one, but the cumulative effect can lead to interesting traps.

You might find yourself in a place where, despite your best intentions and years of wisdom, you and your organization have lost sight of what's most important to your customers. Worse, even when you find out what's important, your processes can be so rigorous that they thwart common sense. You still end up shipping a bad product.

After finding ourselves in just such a position in late 2011, my team and I set out to change everything. We started with the design process and adopted Scenario-Focused Engineering. Then we changed our processes to become more agile, adopting a team-wide cadence of four-week cycles with individual groups measuring their work day to day. Finally, we changed our organizational structure to eliminate artificial specialization of the development and test teams.

Figuring out what to do and implementing it across an organization of a thousand plus people took nearly a year. Mastering the new way of doing things took another 18 months on top of that. We had incremental gains at every step, but no one was certain that the new way was better than the old until near the end of the process. Then it became obvious that we were producing better products in less time and with less wasted effort.

At every step there was reluctance and even resistance, sometimes within the team, but just as often from outside—other organizations had a vested interest in defending the old way, and sometimes management feared what was new.

Perseverance is required if you intend to implement change at scale. Fixing products is easy. Fixing the processes and organizations that build them is hard. The tools and techniques in this book are a powerful starting point for change. Use them wisely on your teams.

Hal Howard
Corporate Vice President
Microsoft Corporation

This page intentionally left blank

Introduction

We are all too busy these days to do anything that doesn't add value to our lives in one way or another. So why are you spending your precious time with this book? Did somebody recommend it to you? Did you flip through the pages and see something that resonated with you? What added value are you expecting this book to provide?

If we've done our jobs well, the answer is simple. Circumstances in your life right now are urging you to improve your craft as a software engineer. You may want to improve the products and services you create for your customers. Perhaps you've noticed that your competitors are delivering solutions that seem to resonate with customers better than yours do. You may be thinking that there's a better way for your team to develop products, and you're looking to be a change agent for your team. Or you may be dreaming of creating the next breakthrough product—an offering that becomes so loved that it will have its own fan club. You aspire to get closer to your customers' needs and desires and to build products that they crave, products they use because they want to, not because they have to.

We think there is a better way to engineer software. We believe that innovation and creativity can be taught, and that there is science behind it, not just art. This book bridges the gap between the power of analytical, deductive reasoning and the seemingly unstructured world of creativity, innovation, and customer delight. It presents both the methods (things you do) and the mindsets (attitudes and points of view) that have been shown over time to be effective tools for creating desirable, innovative products.

Who are you?

You are an engineer. You think logically and systematically. You're smart, and you don't have a lot of patience for those who aren't. You value rigor, science, and efficiency. You strive toward finding elegant solutions to every problem. You like to get things done, to make visible progress, to cross items off your to-do list. You don't like to waste time, and you *hate* rework. And you certainly don't want to waste any more of your life writing code that no one appreciates.

You'd like to be more creative, to be more innovative, to do something that has never been done before, that no one has ever imagined. You aspire to invent something so groundbreaking that it will change the world. But all the things you've read about innovation and creativity feel a bit hollow to you. You sense a lot of hand waving and magic pixie dust and see no real substance; there's too much art, and too little science. But still, you have a nagging feeling that there has got to be a better way and that your current approaches aren't quite working as well as you'd like. This book is for you.

Or perhaps you lead a team of software engineers and want to shift the focus of your team from caring about the innovative details of the technology you're building to caring more about delighting the customers you want to serve. This book is especially for you.

Or maybe you're a designer, a user researcher, a product planner, a project manager, or a marketer and work with engineers who tend to influence big decisions on the basis of their technical knowledge. You wish those engineers knew better how to understand and use your expertise. This book is for you, too.

Our story

We are both engineers, who together have more than 35 years of experience designing, building, and shipping software products and online services at Microsoft. During our careers, we've witnessed a few teams that had great success in building innovative products that resonated with customers. These teams had a few things in common: a strong vision, a deep sense of empathy for their customers, and an iterative approach, sometimes trying tens or even a hundred alternatives over the course of a single release. One other common trait emerged—a healthy and productive relationship between the engineering team and the user-experience design and research team. Instead of working separately, with one team tossing ideas and designs over the wall for the other team to build, the engineering and design teams began to function as an integrated whole. Indeed, the very qualities that made those engineering teams so strong—their iterative approach and customer focus—were also core values of their user-experience partners.

With firsthand experience in using a highly iterative, customer-focused approach on several of our own projects—as well as living through some less-than-ideal situations—we knew how powerful this approach could be in developing outstanding products. Yet we also saw that some of our fellow engineers found this approach counterintuitive. Where we saw tremendous quality and efficiency in getting regular feedback from customers, we were surprised that others feared wasted effort. While some people saw long lists of features as a structured way to plan work, we feared that actual customer needs were being forgotten. Where we saw a rigor and science behind a customer-focused, iterative approach, some people saw only the artistic skills of craftspeople who knew how to make beautiful graphics. Both perspectives had their merits, but few teams found a middle ground to work from.

We also noticed that we weren't alone. A large number of engineers—as well as designers, researchers, marketers, and business people—also saw power in a customer-focused, iterative approach. Some of these people found one another and thrived as they helped their teams build innovative, groundbreaking products. However, many of these people felt increasingly alone on their teams—lone wolves unable to convince their pack to change its technology-focused approach. Most team members knew that they should be talking to customers more often, gathering more feedback, caring more about what customers thought, and striving for a higher level of quality, but the rhythm of daily work just didn't leave much room for this to happen.

We founded the Scenario-Focused Engineering (SFE) initiative in 2008 to help close this gap. On one side, we sought to help every engineer see the value of using a more customer-focused, iterative approach to developing software and realize that this approach is fundamentally logical, rigorous, and based on science. On the other side, we wanted to enlist and empower those who already understood these ideas and help them become leaders on their teams as they blazed a path to the future. It was a grassroots effort aimed at a pretty lofty goal: to fundamentally rethink how engineering teams go about designing products, services, and devices and to drive company-wide change. We were lucky to get to focus on this mission as our full-time jobs, as part of the Engineering Excellence team within Microsoft's Trustworthy Computing division. Over the next six years, we reached more than 22,000 engineers, and in the process learned a lot about what practices work in real life and how to catalyze significant change on a large scale.

What we discovered

We spent the first months conducting lots of interviews across the company. We had some hypotheses about the issues teams were facing, but we wanted to validate our assumptions by talking to lots of engineers on different product teams. The problems weren't difficult to identify, and we quickly noticed that the same stories were repeated over and over again:

> *Each person on the team has a different idea about what's most important, what we're building, who we're building for.*
>
> *We do usability testing and betas, but the feedback comes back too late in the cycle to address it.*
>
> *We ended up having to make a huge design change late in the cycle and had to throw away a lot of code. We wasted a lot of effort.*
>
> *We had more features, but the competitor ate our lunch because their product had a better overall experience.*

After a while, we began asking every team we worked with whether these statements sounded familiar. Their response was yes. But almost universally, very few teams had made any substantial progress in fixing these issues or even knowing where to start.

At the same time, the industry was rapidly accelerating toward online services, and with that shift came a desire for more frequent releases, even multiple releases each day, to provide new functionality to users as quickly as possible or to fix bugs as soon as they were found, not weeks or months later. This sea change intensified customer expectations, and we heard a growing voice within teams that we needed to find more efficient, effective ways to ship the right stuff, faster.

We already had the intuition that techniques such as the user-centered design methodology were an important missing link. Indeed, we weren't the first ones to discover this. Microsoft already had a long-running training course, offered every few weeks, that took students through the basic principles and techniques of user-centered design. The class was exhilarating to teach. Frequently, some students had breakthroughs during class and left feeling passionate and inspired. Yet, later on,

we regularly heard from many of these same students that they weren't able to put any of the ideas they learned into practice. Between their manager, coworkers, project schedules, tools, performance-review commitments, and all the other realities of their work, these new approaches simply didn't fit.

Other students simply didn't understand what was being taught and occasionally became troublesome in class. These cases were rare (skeptics usually don't elect to take an optional course on a topic they don't believe in), but we certainly did get a taste of the depth of skepticism that could be found in our engineering community.

It became clear to us that to really light a fire would require much more than having individuals take a one-day class. We doubled down on our mission to catalyze change within the company and identified two central insights that drove our work for the next six years. First, while engineers are known for being intelligent and adept at solving complex problems, they tend to view the world differently from people in creative disciplines. A large gap exists in how these two groups naturally approach solving problems, in the processes they use, and in the mindsets they maintain. For example, imagine a designer at a creative brainstorming session. She embraces the most unusual ideas and encourages more of them, without restraint and regardless of any practical implications. She's comfortable following somewhat unpredictable paths of reasoning and has faith that the process will result in discovering a potentially innovative solution. Now envision an engineer at this same brainstorming session. He will likely look at the list of ideas and begin to prioritize them based on criteria such as market readiness, the cost of delivery, or the amount of effort required. He might ask for clarification of the problem statement and begin to do a root-cause analysis. The engineer will immediately begin to start *narrowing* the possibilities to get to an answer, but the designer will strive to *broaden* the possibilities. We believe that to optimize the chances for success, you need both approaches (or styles) to balance and complement each other. We've observed that most teams tend to strongly favor one style over the other, and at Microsoft that style almost always favored the engineer.

Second, to get any of our ideas adopted was going to require a whole team—a systemic effort that started with each team's leaders and moved down to every individual contributor. The reality is that it takes a team to build a product, and if you expect change to occur, you need to push the entire team toward change. Customer-focused, iterative product development is not something that can be delegated to someone else to do. It takes the entire team, especially those engineers whose first response to the ideas of SFE is deeply skeptical.

Where did SFE come from?

The general principles behind SFE are not new. They come from sources such as user-centered design (UCD), design thinking, customer-focused design (CFD), human-computer interaction (HCI), and practices we observed while working with Microsoft teams and from the industry. (Iteration and customer-centricity are both very hot ideas right now, and have only gotten hotter since we started teaching SFE in 2008. You can find many great books on these topics; we've listed some in Appendix C.)

If we had to choose the single largest influence behind SFE, it would be the ideas and practices of design thinking, a holistic method of problem solving that is based on the practices of user-centered design. It is not a stretch to say that SFE is an instantiation of design thinking wrapped in engineering terms and concepts.

However, during our journey of creating and refining Scenario-Focused Engineering, we were delighted to see other new approaches to software development gain traction and popularity. As SFE was first gaining momentum within Microsoft, Agile development was quickly becoming a mainstream practice. As we learned more about Agile and saw how compatible it was with SFE, we made some adjustments in the way we spoke about SFE to find the most natural way to present its concepts and practices in the context of Agile development. That adjustment came quite naturally because SFE also values a highly iterative process, and as such is well suited for teams that have already embraced Agile development practices. Throughout the book we mention Agile concepts and terminology and point out ways to incorporate SFE techniques into an Agile development environment.

In 2011, Eric Ries published *The Lean Startup,* and the ideas he describes have since gained tremendous popularity. *The Lean Startup* is also based on the notion that you want to learn quickly by trying out ideas. Ries's approach is generally focused on using actual customer behavior to determine the viability of the business ideas you are pursuing. It has a strong focus on experimentation and making adjustments based on what you learn. In Chapter 4, "Identifying the target customer," we discuss the importance of identifying specific target customers and honing in on a viable business strategy. *The Lean Startup* provides a wealth of ideas, stories, and examples for achieving these goals in ways that are highly synergistic with the activities SFE offers for thinking through complete end-to-end customer experiences.

About the SFE workshop

This book attempts to relay the content of our SFE workshop as well as the experience that a participant might have. It also passes along insights we've gained after working with teams on implementing SFE over the past six years, so it goes into much more detail than the original workshop.

There was no executive mandate at Microsoft for teams to go through SFE training. Every team engagement was driven at the grassroots level, and teams sought SFE training based on word-of-mouth reports. Leaders made an informed decision for their team, when and if the timing was right, which was often at the start of a major release cycle. Sometimes the engagement was scoped to an individual team, but it often involved an entire division of several hundred or several thousand people.

It's helpful to know a bit about how we run the workshop. Most importantly, the workshop is a team event, and the involvement of management is required before, during, and after the workshop. A single instance of the workshop is made up of four mandatory sessions:

- **Senior leader day** This is an all-day session with the highest-level leaders of a team (vice presidents, general managers, development managers, and so on). This session typically involves 10–35 people. We present the content of the workshop and discuss implications for

the leaders' organization. We also describe any customization of the workshop that's desired. At the end of this session, a decision is made whether to pursue training for the larger team or division.

- **Preworkshop meeting for group managers** In separate one-hour meetings, we preview the workshop for the three to five group-level managers on each individual engineering team. This session is required for each team. We discuss the expectations of the management team and the logistics for the daylong workshop. We've learned that even for very large organizations, it is crucial to get buy-in from managers for each engineering team separately.
- **The workshop itself** In a full-day, hands-on workshop, all members of each engineering team (and often key partners) are required to participate, including group managers, developers, testers, program managers, user-experience professionals, product planners, etc. We optimize the room, furniture layout, and training curriculum for a class size that accommodates a full engineering team of up to 120 people. By training together in the same room, team members can look around to see that the entire team is present and that leaders are fully engaged. When the energy and buzz of the room escalates and a promising change or action item for the team is discussed, the seeds for consensus are planted in real time.
- **Postworkshop meeting for group managers** We meet with each set of group managers a second time for two to three hours. We analyze and discuss feedback generated by their team during the team workshop. This "crowd-sourced" feedback gives the managers insights into which ideas are most relevant for their team, helping them settle on an initial set of techniques to adopt and put a plan in motion.

The hands-on exercises and content of the workshop are customized for each team's specific target customers, development practices, and market situation. Throughout this book, we introduce some of these exercises, with the intent to give you a sense of both the personal experience and the team dynamics that take place. You'll gain value from doing some of the exercises on your own, and others can serve as interesting thought experiments.

We have delivered the SFE workshop to hundreds of teams and worked alongside many of them as they put SFE concepts into practice. We've learned a lot, fine-tuned ideas that work, and abandoned those that don't. We've discovered many gotchas and pitfalls, and we've had the opportunity to observe and participate in large-scale change in teams of all shapes and sizes. Some teams we worked with made huge strides with the ideas in this book, and we'll share some of their stories and successes. However, we've also been humbled to see that some teams were not able to make the shift to adopt many of the ideas in this book, despite (in most cases) valiant efforts. The places they stumbled are noteworthy, provide important lessons, and are a reminder to future teams to take the change process seriously.

The workshop, however, is just the beginning of the journey. It serves as a catalyst to get the whole team on the same page, working with a common vocabulary and aligned to similar aspirations. The real work begins the day after the workshop, when the team must start to put these ideas into practice by adjusting tools, schedules, and roles and building new habits. Although the ideas in this book are straightforward, for most software engineering teams, they represent significant changes to

day-to-day work habits and practices, as well as team culture, values, and habits. The change process is significant, and we'll discuss aspects throughout the book and in detail in Part III.

What will you get out of this book?

This book is about discovering how to put customers at the center of your engineering efforts. It describes effective and efficient ways to iterate and explore so that you can deliver solutions that customers will find deeply delightful, and deliver them as quickly and efficiently as possible. It's our hope and intent that this book will inspire you to reach out and connect with your customers in a richer way than you ever have before.

We also strive to arm you with enough data, tools, examples, tips, and gotchas that you have the confidence to take action and will be motivated to try out some of these ideas in your current project; in short, after reading this book, you will know just enough to be dangerous. We include references to our favorite books on each topic at the end of this book so that you can dig deeper into the techniques you choose to invest in. We expect that you will need to reach out to experts or to leverage some of the references we list to put these practices to full use. This book will help you figure out the menu of what's available and help you make informed choices about the most appropriate techniques for your individual situation. Finally, we hope that many of you are energized by this book and begin an effort on your team to put customers at the center of your work.

A special note to the user-experience community

Thanks for checking out this book. While you already know that it is not targeted directly at you, there is tremendous value in bridging the gap between what you know how to do and what engineers know how to do, and that is what we are aiming to do. As the saying goes, "It takes two to tango." We are trying to get the dance started.

Over the years, we've noticed that user experience (UX) designers and researchers have a few fairly predictable reactions to the idea of presenting a detailed workshop about customer focus, iteration, and design practices to engineers. We've spoken to other authors who have written books on the design process and have been fascinated to discover they have had the same experience with the UX community. Those reactions are worth a bit of discussion here so that you can bypass the anxiety and get directly to the value. Here are the three gut responses we tend to hear from UX professionals:

We learned all this early on in design school. These are the basics. This isn't even the meat of design work. It's not worth my time; there is nothing here for me to learn.

Hey, wait a minute! You can't just boil down these complicated and sophisticated concepts and skills and wrap them in warm fuzzies about collaboration and expect amateurs to get the same great design results I've produced for years! Sure, readers will learn some principles and buzzwords, but I don't want software engineers thinking they really know this stuff. And you know these engineers: when they learn a little, they think they know everything. I don't want

to have to clean up more mess. I won't admit this publicly, but when it comes to budget crunch time, this will make me worried that the pointy-haired managers will see me as redundant!

We're BFFs at last! Is it possible that by learning more about the importance of the design process, engineers will finally see the value I can bring to the team? If I can be seen as a champion for a customer-centric effort using terminology and processes the team understands, maybe they'll stop throwing decorative UI projects over the fence and will actually engage with me from the product's conception through delivery.

We get it. We understand. We can say with some confidence that if you went to school for interaction design or user research, most of the principles and techniques described here will be well known to you. (To software developers: think about it as giving an "Introduction to Data Structures" class to someone who's never studied computer science.) And we know that we cannot represent the full scope of the research and design professions in a few chapters aimed at people who are trained in different skills.

However, our experience is that if an engineering team embraces the ideas in this book, several outcomes are likely. First, the demand for your skills and experience will increase, not decrease. The engineers will now understand the breadth of your experience and will understand that it's not quite as easy as it looks, and you will become very, very popular. Second, you will spend less time fighting for the right things to happen, as the entire team will be aligned on the customer's needs from the beginning. Third, if it does not exist already, the team will demand that you create and fill a reliable customer feedback pipeline. And finally, you will have more time to go deep on detailed research and design work. Other people on the engineering team will now be able to effectively contribute the basic elements that frequently spread your time too thin. You will be able to build on those elements and proceed with a newfound confidence, finally having the opportunity to use your true expertise. The really cool part is that once you make that shift, everyone on the team will have a shared vocabulary. Everyone is sitting at the same table, speaking the same language, able to work together more effectively than ever to create products that will delight your customers.

Organization of this book

This book is divided into three sections. Part I, "Overview," delves into the reasons why customer delight and end-to-end experiences are so important and introduces the Fast Feedback Cycle, which powers the Scenario-Focused Engineering approach. Part II, "The Fast Feedback Cycle," details each stage in the cycle, describing the principles and key considerations for each stage as well as techniques to choose from. In several "Deep dive" sections, we offer details about our favorite, most broadly useful techniques. Part III, "The day after," discusses insights we've gained from watching many teams adopt the practices of Scenario-Focused Engineering. The insights involve project management implications and the realities of shifting team culture.

This book also includes appendixes that provide resources and references. Appendix A presents the "SFE capability roadmap," a checklist teams can use to gauge their level of sophistication in 10 core capability areas. Appendix B provides a one-page diagram of the Fast Feedback Cycle along with the most essential ideas for each stage. It makes a great poster. Appendix C, "Further reading," provides a list of books and resources for further exploration. Appendix D offers two case studies of teams that used the SFE approach. Appendix E, "Desirability Toolkit," is a reference list of the words used in the Desirability Toolkit technique presented in Chapter 9.

How to read this book

This is a pretty long book, with a lot of varied content. Some sections read like hard-to-put-down stories, perhaps something you'd enjoy reading on an airplane. Other sections present encyclopedia-style descriptions of tools and techniques that are perfect as a reference guide. Still other sections give in-depth instruction on a specific technique that you could apply in your work.

While we certainly welcome you to read this book cover to cover, we know that different people have different needs from a book such as this one. We've organized the book so that it works for several different kinds of readers and situations. Here are a few suggestions for ways you might experience the content of this book:

- **The toe in the water** You've always been interested in innovation, technology, creativity, and building products that customers' love. You suspect that you can do some things in your own work that would make a big difference. But you are super busy, don't have a lot of time to read, and certainly don't want to take on the task of plowing through a long, detailed reference. You'd love to take a couple of hours and learn some new things and perhaps feel energized to bring up a couple of new ideas at work.

  Read Chapters 1 through 3. These are fun, easy chapters and stand alone quite well. You'll get a great introduction to the ideas behind SFE—why it's important and what is involved—and come away with some good anecdotes to share at the water cooler.

- **The deeper read** You are about to get on a coast-to-coast flight, and you're looking at the bestsellers in the airport bookstore. You're looking for something that will be an engaging read, that has the potential to add a lot of value to your work when you return. Wondering if this book might be helpful, you flip through a few pages and find yourself saying, "So true, so true. Oh, my gosh, that's my team," and your curiosity is piqued.

  Read Chapters 1-3 and the first half of each chapter in Part II, skipping the tools and techniques sections. Reading these portions of the book will give you a solid overview of SFE and the Fast Feedback Cycle, including the standout mindshifts and behaviors. If you have more time, take a look at Part III and Appendix A to crystallize your thoughts around the current state of your team, a few things you might like to try, and how you might go about doing so.

- **The student** You're in the middle of a university program, working toward a degree in computer science. You're taking a hot new class that claims to bridge the gap between the business, computer science, and design schools and this is the textbook.

  Focus your reading on Part II, Chapters 4-10, which describes the Fast Feedback Cycle and the techniques in detail. Parts I and III and the appendixes provide some real-world context, but the meat of the book is in Part II and includes all the how-to information and the specific techniques you'll draw from. We hope you have fun and that your class project yields a bazillion-dollar idea. Go forth and delight!

- **The change agent** You are already a fan of SFE and the Fast Feedback Cycle. Maybe you've taken a class, have used these ideas on a previous team, or have read through Part I and gotten inspired. Now you want to help your team gain proficiency in some of these practices.

  Start by reading Part III (Chapters 11 and 12) to get some perspective on how best to introduce a new practice to your team. Then, pick a chapter in Part II that covers an approach that you think will add the most immediate value to your team. Focus your team on trying out a few of the different tools and techniques presented in that chapter and gain some experience in that area. After some early success, you and a few of your cohorts might dig into the SFE capability roadmap in Appendix A to assess where you are and to figure out what areas you might work on next.

Contact us

SFE has grown beyond Microsoft. There is a large and growing community of SFE practitioners worldwide. Join in the conversation. We love to hear from readers. Please tell us how these ideas are working for you and what new techniques you've developed so that we can all continue to learn and iterate: http://www.scenariofocusedengineering.com.

Acknowledgments

Although this book was largely written by two people, it is the reflection of the hard work, passion, and dedication of hundreds of people—the Scenario-Focused Engineering community, whose members spent countless hours teaching, iterating, coaching, and practicing these techniques with their teams throughout the past six years.

First and foremost, we'd like to thank our families.

Austina: To Erik, Maja, and Viktor, thank you for your infinite patience, allowing me to sit in my chair and write—for what seemed like ages, and long into the wee hours. An enormous thank you to my parents, who lived with us this past year and helped so much with keeping the household running, feeding all of us, and delivering the kids wherever they needed to be. Thank you all for being there and for believing in me and in this crazy project.

Drew: To Kristy, Kylie, and Derek. I am so lucky to have each of you in my life. You have been my inspiration and my muse. Thank you for understanding when I didn't have time to cook a proper dinner (although I have learned how much you do like the frozen Chinese food from TJ's), was late picking you up from school, and why I was occasionally grumpy and tired after a long day of very deep thinking. But most, thank you for being you and for sharing your lives with me. I love you all dearly.

A handful of people have been instrumental in helping us write this book. Without them, this book would never have been finished. Thank you to Kent Sullivan for being our resident UX research guru and for providing data, countless interviews, and tireless reviews and for essentially ghost writing Chapter 5 and Chapter 9. Thank you to Norman Furlong, who was our official sidebar wrangler and who kept our spirits high. Many thanks to William Parkhurst, who provided inspiration and motivation, gave valuable feedback every step of the way, and, when we needed it the most, provided surfing lessons. Finally, thanks to Jeanine Spence, who did the lion's share of the work to conceive, build, and test the SFE capability roadmap (see Appendix A) and who synthesized data from more than 60 experts across Microsoft and the industry.

We'd like to offer a special note of gratitude to Dr. Indrė Viskontas, our resident neuroscientist and opera star. When we first approached Indrė, we had one topic in mind that we wanted her thoughts on. That initial idea quickly grew to a long list of topics, as it became clear how much overlap there was between our work and the available science to support it. We are deeply grateful to Indrė for her insights and contributions to this book.

In all, the book contains roughly 40 "SFE in action" sidebars. Thank you to everyone who contributed sidebars or provided their expert advice. Sadly, because of space constraints, we were not able to print all the sidebars that were contributed, but we are deeply thankful to all the people who took the time to suggest sidebars and other content proposals.

Many people offered their time and patience to help us test the concepts in this book and to review the text as it was drafted, rewritten, fine-tuned, edited, rewritten again, and finally sent to production. A special thank you for going above and beyond goes to Paula Bach, Bill Chiles, Steven Clark, Terrell Cox, David Deer, Bernie Duerr, David Duffy, Paul Elrif, Serguei Endrikhovski, Valentina Grigoreanu, Kevin Honeyman, Karl Melder, Susan Mings, Damien Newman, Bruce Nussbaum, William Parkhurst, Victor Robinson, Prashant Sridharan, Bill Storage, Mike Toot, Sam Zaiss, the entire SFE team, and the many other SFE champions within Microsoft.

Throughout the book, we talk about design as a team sport. The team that developed, taught, iterated, and coached SFE is no exception. We are ever grateful and humbled to have had the opportunity to work with the strong group of passionate engineers who made up the SFE team over the years:

- **SFE instructors and core team** Jeanine Spence, the synthesizer-conceptualizer extraordinaire; Kent Sullivan, our ultra-collaborator; Margie Clinton, who still holds the record for teaching the most classes; Ken Zick and Norman Furlong, who made the toolbox repository happen; Alex Blanton, who brought us "The Delighted Customer" blog; Ed Essey, who folded in Agile concepts and crucial platform team examples; Seth Eliot, who taught us about experimentation; Phillip Hunter, who taught us the nuances of the meaning of "delight"; and

Court Crawford, who was such a passionate champ that he joined the team. The early teaching team included Bill Begorre, Bill Hanlon, Richard Kleese, Marina Polishchuk, Alec Ramsey, and Surya Vanka. The international teaching team included Alex Cobb, Sven Hallauer, Antonio Palacios, Lior Moshaiov, and Jayashree Venkataraman. The early PM team was John Pennock, Li Lu-Porter, and Van Van. The founding team included Keith Bentley, Margie Clinton, and Michael Corning.

- **Instructional design team** Robert Deupree, Fredrika Sprengle, and Brian Turner.
- **Operations team** Robyn Brown, Joetta Bell, Kim Hargraves, Cristina Knecht, and LouAn Williams.

Thanks to Wendy Tapper for helping to get SFE off the ground by designing the "green card" in her spare time; our calling card has truly stood the test of time. Thanks also to Peter Moon, who saw the value in SFE early on and customized and adapted the SFE workshop for delivery to several thousand engineers in the IT departments at Microsoft. Gratitude also goes to Karl Haberl, Martin Plourde, and Brian Pulliam, who taught us how to measure and scorecard SFE's impact. Special thanks to Alec Ramsey for conceptualizing the first versions of the Fast Feedback Cycle model, which became the backbone of Scenario-Focused Engineering. Thanks also to Dean O'Neill, who taught us the value of good tooling and independently spearheaded the Microsoft Process Template in Team Foundation Server, which enabled SFE work-item tracking for many teams. We'd also like to thank a handful of expert Agile practitioners who helped us over the years: Bill Begorre, Arlo Belshee, Ed Essey, Sam Guckenheimer, Bill Hanlon, and Scot Kelly.

As we were developing the SFE program, Irada Sadykova and Eric Brechner were the leaders who created the space and budget in which we could operate. They battled for our cause in the metaphorical executive washrooms when SFE's success, popularity, and value were not immediately obvious to those who needed to care. Were it not for their hard work, we would never have gotten this project off the ground. More recently, thanks to Peter Loforte and Debbie Thiel for continuing to support SFE and especially for enabling the conditions to allow this book to be written. Thank you also to all our colleagues in Engineering Excellence for their support along the way. A special thank you goes to Surya Vanka, who represented SFE to the UX leadership community, served as our design guru, and helped recruit incredible talent to the SFE team.

Our first champion at Microsoft was Matt Kotler, who was instrumental in bringing the SFE workshop and ideas to the entire Microsoft Office team. Thank you, Matt, for being the first to believe and the first to champion SFE throughout such a large organization. Similarly, Ian Todd had a huge impact in making these ideas take root in Windows Phone. Susan Mings and Dean O'Neill had a similar impact in Windows Server, as did Lisa Mueller in Dynamics and Tracey Trewin in the Developer Division.

Throughout the years, a large and thriving community of SFE champions and change agents has developed, and each of them deserves big kudos for their work, passion, and perseverance: Bia Ambrosa, Gabe Aul, Paula Bach, Cyrus Balsara, Don Barnett, Richard Barnwell, Dan Barritt, Tom Baxter, Derrick Bazlen, Laura Bergstrom, Brijesh Bhatia, Safiya Bhojawala, Jeff Braaten, Tim Briggs, Adam Bronsther, Graham Bury, Jeremy Bye, John Cable, Ben Canning, Greg Chapman, Alison Clark,

Steven Clarke, Ken Coleman, Jeff Comstock, Matthew Cosner, Robin Counts, Clint Covington, Arne de Booij, Lance Delano, Shikha Desai, Tammy Dietz, Serguei Endrikhovski, Umer Farooq, Rob Farrow, Tricia Fejfar, James Fiduccia, Joseph Figueroa, Ned Friend, Bob Fries, Jim Fullmer, Jean Gabarra, Tyler Gibson, Stephen Giff, Valentina Grigoreanu, Carol Grojean, Sam Guckenheimer, Joe Hallock, Mark Hansen, Ed Harris, Geoff Harris, Steve Herbst, Steve Hoberecht, Kevin Honeyman, Christy Hughes Harder, Jeremy Jobling, Joe Kennebec, Alma Kharrat, Ruth Kikin-Gil, J. T. Kimbell, Bernhard Kohlmeier, Miki Konno, Kevin Lane, Sue Larson, Mikal Lewis, John Licata, Jane Liles, Ulzi Lobo, Derek Luhn, Craig Maitlen, Steve May, Michael McCormack, Ford McKinstry, Soni Meinke, Karl Melder, Trish Miner, Becky Morley, Cathy Moya, Lisa Mueller, Joe Munko, Mark Mydland, Dean O'Neill, Susan Palmer, Sheri Panabaker, Sachin Panvalkar, Milan Patel, Mike Pell, Ken Perilman, Nancy Perry, Mike Pietraszak, Barton Place, Chandra Prasad, Ed Price, TJ Rhoades, Lawrence Ripsher, Ramon Romero, Dona Sarkar, Joel Schaeffer, Ravi Shanker, Wenqi Shen, Jasdeep Singha, Shilpi Sinha, Cameron Skinner, Bill Stauber, Derik Stenerson, Christina Storm, Philip Su, Deannah Templeton, Mike Tholfsen, Robin Troy, Jonathan V. Smith, Kimberly Walters, Kim Wilton, Sam Zaiss, Brant Zwiefel, and so many more.

Thank you to all of the teams across Microsoft that put their trust in us to show them some new tricks. Special thanks to the entire Office team for living through our first big iterations of the workshop. It was a pretty rocky road in some regards, but we and your leaders learned a lot, SFE is now better for it, and Office has forged a great path forward. Thank you to the Dynamics team, which was the first large organization to figure out how to integrate SFE practices in an Agile environment at scale—and to demonstrate unquestionable business results from their investment. Thank you to Windows Phone for doubling down on SFE several times over the years and leading the way on the importance of brand, and to Windows Server for going the distance in building tools and infrastructure to support SFE. These and many other teams were the reason we continued investing in this work.

At Microsoft, senior leaders of user experience teams gather monthly as the User Experience Leadership Team (UXLT). That team provided valuable support, resources, encouragement, and course corrections throughout our journey. Thank you UXLT! We'd like to specifically call out appreciation to Lisa Anderson, Tom Bouchard, Andy Cargile, Terrell Cox, Monty Hammontree, Steve Kaneko, Laura Kern, Kent Lowry, Kartik Mithal, and Jakob Nielsen for their sponsorship.

One of the most rewarding aspects of driving the SFE initiative at Microsoft was that we developed close relationships with many senior leaders who provided mentorship for us as they brought SFE to their organizations. Thank you all so much for the invaluable contributions you've made to SFE and to both of us—Stuart Ashmun, Joe Belfiore, Erin Chapple, Andy Erlandson, Chuck Friedman, PJ Hough, Hal Howard, Bill Laing, Chris Pratley, Tara Roth, Zig Serafin, and Jeffrey Snover.

Finally, thank you to our developmental editor, Devon Musgrave, and to John Pierce and Rob Nance for their editorial and production work. We appreciate the patience they showed dealing with all of our funky, late, and ongoing requests. Thank you all for helping us make our dream happen!

Errata, updates, & book support

Microsoft Press has made every effort to ensure the accuracy of this book. If you discover an error, please submit it to us via mspinput@microsoft.com. You can also reach the Microsoft Press Book Support team for other support via the same alias. Please note that product support for Microsoft software and hardware is not offered through this address. For help with Microsoft software or hardware, go to *http://support.microsoft.com*.

Free ebooks from Microsoft Press

From technical overviews to in-depth information on special topics, the free ebooks from Microsoft Press cover a wide range of topics. These ebooks are available in PDF, EPUB, and Mobi for Kindle formats, ready for you to download at:

http://aka.ms/mspressfree

Check back often to see what is new!

We want to hear from you

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at:

http://aka.ms/tellpress

We know you're busy, so we've kept it short with just a few questions. Your answers go directly to the editors at Microsoft Press. (No personal information will be requested.) Thanks in advance for your input!

And let's keep the conversation going! We're on Twitter: *http://twitter.com/MicrosoftPress*


SCIENTIFIC METHOD
Keep iterating
NOT DECIDING, DISCOVERING
AGILE
Understand vs. create
Plan to get it wrong the first time
frame
iterate
FAST FEEDBACK CYCLE
Take an experimental approach
Learning by making
scenarios
3 iterations before finalizing the plan
Framing the problem
Due diligence of generating multiple alternatives
User-centered design
brainstorm
build
Observe customers
IDENTIFY TARGET CUSTOMERS
Get feedback early
The Lean Startup
Make a hypothesis
Rhythm of iteration
science of iteration
GET CUSTOMER FEEDBACK

CHAPTER 3

Take an experimental approach

To stand out in today's mature software market, you need to delight customers at an emotional level. A reliable way to delight customers is to deliver an end-to-end experience that solves a complete customer need, even if that means delivering less functionality overall. But how do you build those end-to-end experiences? And perhaps more importantly, how do you know which end-to-end experiences to build in the first place? The secret is staying focused on customers' real-world needs and desires and taking an iterative, experimental approach to zero in on great solutions for those needs.

In this chapter, we give you an overview of the customer-focused, iterative approach that we call the Fast Feedback Cycle. You will see what it looks like, what the basic activities are at each stage, and how the stages fit together. Subsequent chapters dive into more details and techniques for the work you do in each stage of the Fast Feedback Cycle.

Designing a new mouse

Let's start with an example of using the Fast Feedback Cycle to build a hardware device—a mouse. This is a case study of Project Bentley, a Microsoft hardware project that was chartered to build a highly ergonomic mouse.

The inspiration for the mouse was simple—when observing customers who were using Microsoft's ergonomic keyboards, the hardware team noticed that many customers used gel pads or other accessories to make their mice more comfortable to use.[1] A highly ergonomic mouse would be a natural extension to Microsoft's line of strong-selling ergonomic keyboards, and so the project was born.

With a bit of research about situations in which people used a mouse, which buttons and sliders got the most use, and the size of the average hand; a decision to focus exclusively on right-handed users; and a long history of designing mice and other hardware, the team began to brainstorm possible solutions. Here's what its first round of ideas looked like:

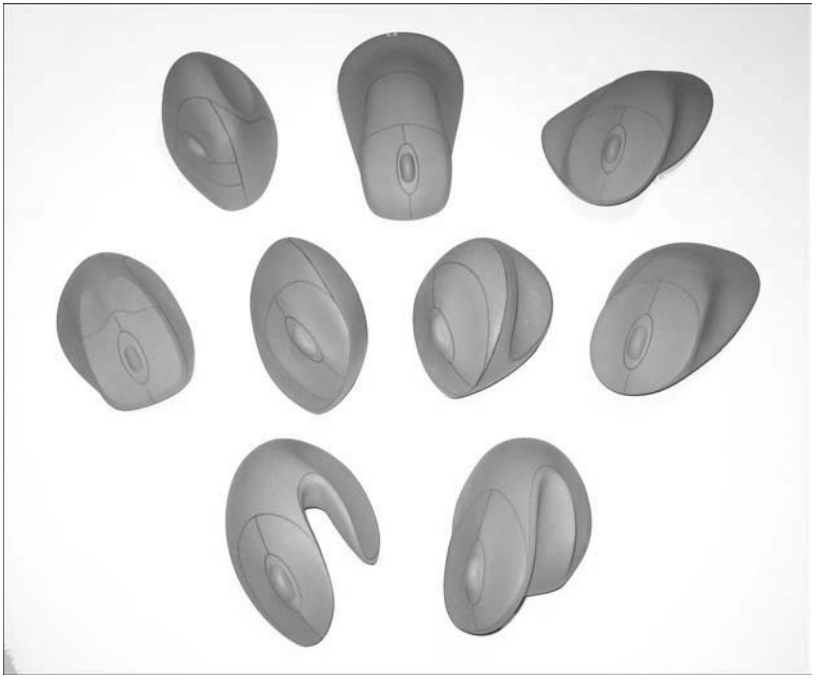
The team made many quick prototypes—about 50 of them in all. Each of the prototypes was made very quickly from inexpensive, easy-to-work modeling materials. None took more than 15 minutes to create, some much less. But take a closer look; many of the prototypes are not finished. In fact, some of them are downright strange. For instance, look at the gray one in the center: Which way do you hold it? Where do the buttons go? Look at the tall one just above the gray one and to the left—it's sharp on top. Would anyone ever ship a mouse that's sharp on top? Many of the mockups look like they were abandoned halfway through. Several have thin lips on the bottom that look destined to crack off and would never pass a manufacturing review.

The point is that the team tried out ideas that even in their mind's eye were going nowhere, just to see whether they might lead to a better idea.

At this stage in the project, it's cheap and fast to try out a new approach, so the team considered as many different shapes, configurations, and form factors that they could think of. *This is a classic brainstorming step, where you cast the net as wide as possible at the very beginning of the process, when your brain is most capable of generating many different ideas.* It turns out that some solid neuroscience lies behind why your brain is much more capable of generating lots of ideas when you brainstorm early, as the first step of a project, before you become too mentally committed to any single approach. We'll go into this in detail in Chapter 7, "Brainstorming alternatives."

To get some input as to which models were working best ergonomically, the team then showed them to a few customers to touch, feel, and hold. The team also began thinking through which approaches were most likely to be technically feasible. It's notable that this first round of customer feedback happened just a few weeks into the project.

After considering that feedback, they produced their second round of ideas:



Note that the team didn't choose just one of their initial ideas to work with in more detail—they were still working on nine alternatives in this round. But this time, instead of using foam and clay, they built CAD models for each mouse and "printed" them out using a 3-D printer to create the physical forms. At this point the details were starting to get worked out. The buttons and precise contours were all there, and you can see that each one now incorporates a scroll wheel, which turned out to be a key request from customers.

The team was now also considering the technical implications of each design. Would it work for manufacturing? Would all the gearing and components fit inside the case? What kind of plastics could be used? In parallel, they continued testing with users to get feedback about how the mouse felt in people's hands—because, after all, the ultimate goal was to build a mouse with superior ergonomics.

Here's what they produced for round three:

Again, the team didn't pick just one concept to move forward with. This time they selected four options to prototype more fully—now with functional buttons, real materials, and all the internal mechanisms. Just as before, they debated the technical feasibility of each design and had customers use these mice in real situations to get detailed feedback about what worked and what didn't.

In the end, here is what they finally shipped, the Microsoft Natural Mouse 6000, released in 2008:

Did you notice that the mouse they shipped is not the same as any of the final four prototypes? While it is most similar to H, look closely and you'll see that it incorporates aspects of all four of the final models. The same sort of combinatoric mixing of good ideas from multiple lines of thinking happened at every stage of this process. Go back and look at the 3-D models—none of them is exactly the same as any of the original foam models. Similarly, none of the four functional prototypes is the same as any one of the 3-D models. As the team iterated, it combined and recombined the best ideas from different prototypes to narrow in on the combination that worked the best—both for technical feasibility and for end-user ergonomics. In the end, they delivered a product that did well in the market, and really delighted their customers.[2]

Engineers naturally tend to iterate ideas. As you work through the issues and get feedback from others on the team, your solutions steadily get better and more refined. However, unlike in this example, you typically start with only one seed—one reasonably good idea of how to solve the problem, and you iterate from there, continually refining that idea until you get to a final solution.

However, if you think back to your mathematics background, starting with one seed is a really good way to find a local maximum in a complex plane. If you want a more statistically reliable way to find the global maximum, you need to start with more seeds. This is the magic behind the iterative approach illustrated by the mouse example—combining and recombining the best ideas from multiple lines of thinking within the Fast Feedback Cycle to give you the very best odds of finding the most optimal solution across all of your constraints.

Some of you may question whether this illustration is even relevant to software development. We chose this example because it provides a great visualization of what an ideal iterative process might look like. It's a good example precisely because it is so physical and easy to photograph and view step by step. For software projects, you don't prototype with clay and foam, but on paper, with wireframes, whiteboard drawings, storyboards, PowerPoint mockups, prototyping toolkits, flow diagrams, or even by writing prototype code. To capture a similar snapshot of the iterative stages in a software project would take stacks and stacks of paper, and the patterns would be much harder to see at a glance. But regardless of the form factor, the core ideas are exactly the same:

- Start with a clear idea of which target customers you are building for.
- Understand those customers' needs and desires in the context of their real-life situations.
- Explore many possible ideas, especially in visual ways.
- Build several rapid prototypes of the most promising ideas.
- Evaluate prototypes with customers to get feedback and learn, while working through technical feasibility in parallel.
- Refine prototypes, gradually focusing in on fewer ideas, adding more details at each successive round, and eventually writing production code once your plan has stabilized.
- Repeat as needed.

It's worth taking a short pause to ponder a quick thought experiment. What would it take to actually work with multiple ideas in an iterative approach in your software development process? How different would that be? How close are you to doing these things already in your current team and project? What would the implications be if you recentered your whole engineering system on this approach?

The Fast Feedback Cycle

The Fast Feedback Cycle is the heart of Scenario-Focused Engineering, and there's a clear science and rigor for how to do it well.

As the mouse example showed, you want to take a more experimental approach to building solutions. The key term here is *experimental*. This means that your job as an engineer is less about *deciding* what the product will do and more about *discovering* what is actually going to work in real time, in real usage, with real people, and with real technology constraints.

It's important to see how the different parts of the Fast Feedback Cycle fit together to achieve this goal. Together, the parts of this cycle form a general approach to problem solving; they aren't a specific prescribed tool set. In fact, you could (and should) apply lots of different tools at each stage in the cycle, depending on your situation and how far along you are in your project. We'll talk about the most common ones used for software projects throughout the book, and we'll also mention alternatives for more unusual situations or different domains. However, while you need to pick the most appropriate set of tools for your individual situation, the underlying rhythm and general approach shouldn't change. The science (and the power) of the Fast Feedback Cycle is in this rhythm, illustrated in Figure 3-1.

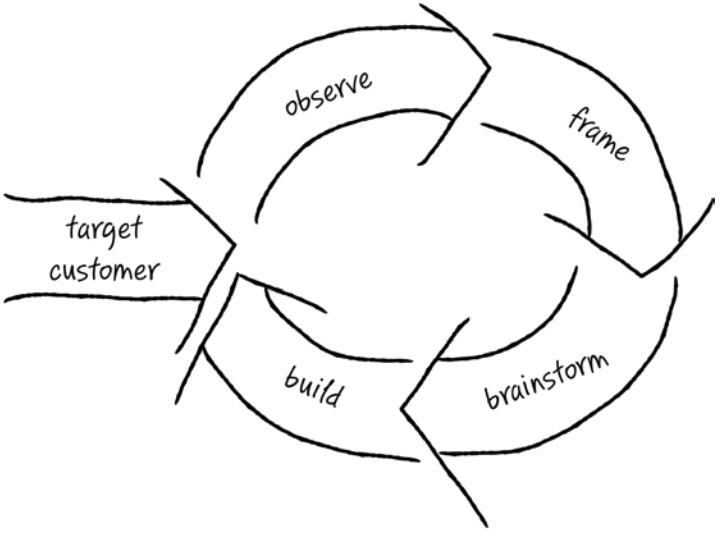
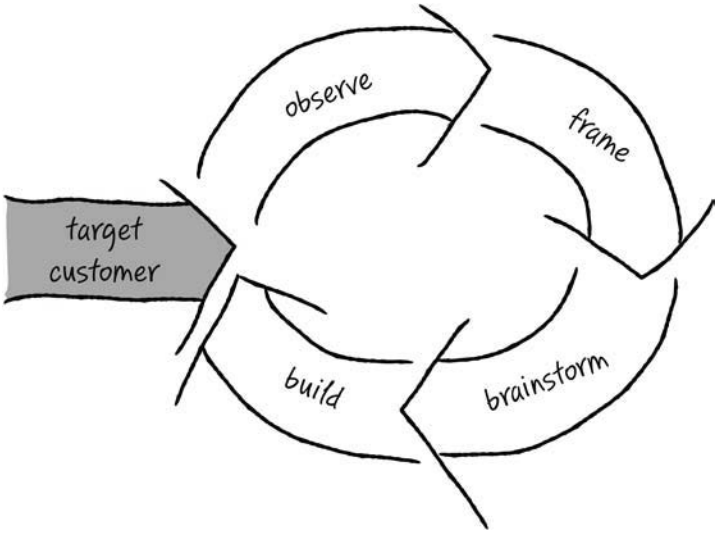


FIGURE 3-1 The Fast Feedback Cycle.

Let's walk through the Fast Feedback Cycle at a high level so that you can see the overall pattern. The chapters in Part II, "The Fast Feedback Cycle," go into each stage in detail.

Target customer

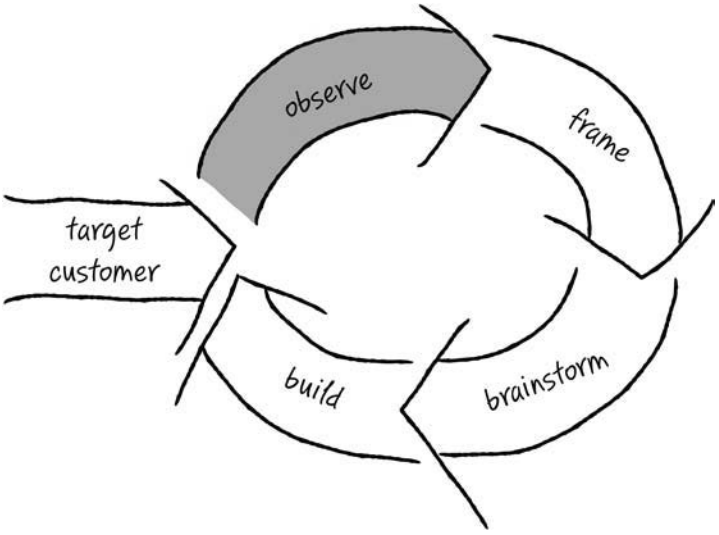


First, before you can start iterating, you need to know who your target customer is. That is, who do you intend to build this solution for? The choice of target customer is primarily a business-strategy decision—which customers are the most lucrative or most leveraged or most likely to drive long-term success?

Defining a target customer is essential because it acts as a lens through which all the other stages are focused. The target customer is the fuel that powers the cycle, and knowing your customer is the most important prerequisite before starting the iterative process.

Chapter 4, "Identifying your target customer," discusses this topic in more depth and describes why this focus is so essential.

Observe



After your target customers are identified, you spend time researching them—looking for their unarticulated (or latent) needs: that is, what customers can't quite tell you themselves but which are the deep desires and underlying needs that drive their behaviors. Identifying an unarticulated need that your competition has not yet noticed, and then building an excellent solution for it, is a great way to achieve differentiation in a crowded market.

Site visits and other observational research approaches help ground you in customers' real-world situations—not in abstractions of what they should need, but in the reality of what they run into in their actual usage. By watching customers in their native habitats (on their couch, in their office, or walking in a crowded shopping mall), you learn things you wouldn't have thought to ask about. For example, "Why are you doing that eight-step workaround when this beautiful feature over here would do it for you? . . . Oh, you knew it was there, but it doesn't work with your company's procurement system? Hmm, you're right; oops, we didn't plan for that situation."

Your goal in observing customers is to gather data that will help you ferret out root causes—not the symptoms or the Band-Aids or the referred pain, but the original source of the problem or the deep human need that is driving the customers' behavior. You may uncover a critical detail—a product interaction you never noticed before. Or you may discover a surprising insight into what really matters for a knowledge worker. Perhaps for that worker deep delight comes not from getting the job done but from unobtrusively staying in touch with his family throughout the workday, which counterintuitively improves productivity because he isn't worried about whether his kids got to school safely.

Collecting and analyzing more numerical, or quantitative, data adds another piece to the puzzle. Whether you are looking at statistics about your competitors or crunching massive amounts of usage data from your existing systems, quantitative data can help alert you to usage patterns, anomalies, or pain points that may help you find a new, unique opportunity to delight your customer.

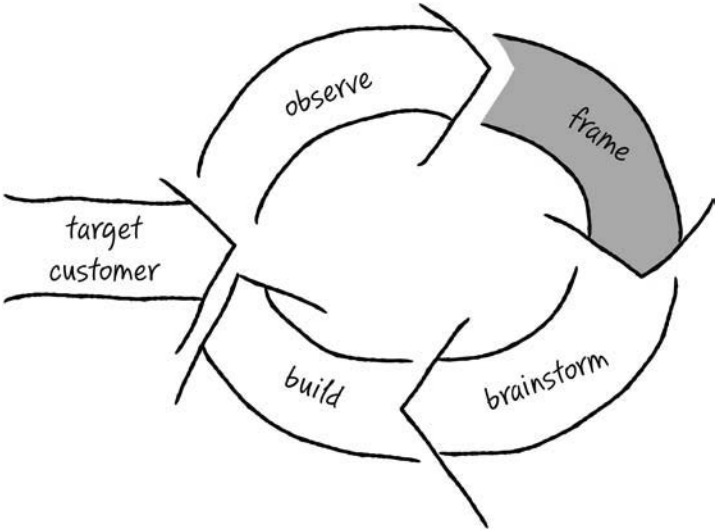
MINDSHIFT

Customers want to have input. Too often there are many layers between the customer and the development team. Any feedback received is often diluted, distorted, and dismissed. Having a direct, real, and personal connection with customers can energize the team, as well as tell your customers that you really care about their needs.

After doing even a moderate amount of research, it's easy to collect a veritable mountain of data points, customer requests, ideas, and notes. These may come in a wide variety of formats: quotes, survey data, photos, video footage, competitive information, usage statistics, and so on. Unfortunately, the breakthrough opportunities you are looking for may not always be immediately obvious. As you get ready to exit this stage, it is vital to look across your research to identify patterns and themes that point to the most pressing customer needs and opportunities. Affinity diagramming is a straightforward and extremely effective technique for making sense of a large amount of unstructured customer data of this sort and will help you identify patterns and insights about your customers that might elude you otherwise.

Chapter 5, "Observing customers: Building empathy," discusses how to use a mix of subjective, objective, qualitative, and quantitative methods to uncover unarticulated customer needs and develop empathy for your customer. It also shows how to use affinity diagrams and other analysis techniques to distill a large set of data points into a small set of key insights.

Frame



While observing customers, you will undoubtedly discover lots and lots of needs and potential opportunities. The next step is to make a judgment about which of those needs are the most important to address, and to precisely articulate what those needs are. This is called *framing the problem*.

Think of it like holding up an empty picture frame in front of an imaginary tapestry that depicts every aspect of your customers' needs, usage patterns, and desires. Depending on where you hold that picture frame, you determine which aspect of the customers' experience you are going to focus on. Will you hold it close and focus in on a specific, narrow situation? Or will you hold it farther away and zoom out to include a complete cradle-to-grave experience? Will you focus on this problem or on that one? On this situation or that other one?

Once you decide on a frame, this allows you to focus on how to fill out the inside of that frame—and not become distracted or continually debate why you are working on this particular need. In addition, framing encourages a bit of rigor to ensure that you articulate exactly which problem or situation you are trying to solve before launching into building solutions, which is vital for alignment when you have multiple people (or multiple teams) who need to contribute to a project.

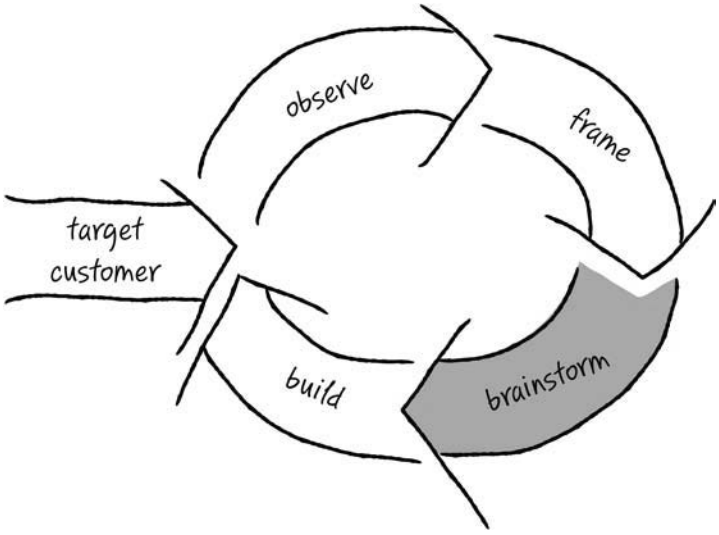
One very helpful and broadly applicable framing technique is to write stories such as scenarios, epics, or user journeys that tell the story of what a customer is trying to accomplish that your product could help with. We have found the scenario technique to be a particularly effective tool for describing the kind of end-to-end experiences that we aspire to build with software. A scenario introduces the target customer, describes a real situation in which the customer has a need, and articulates what qualities a good solution would have—without saying anything about exactly how that solution would work.

Additionally, it is critical at this stage to establish a few metrics to track as key criteria for success and to embed these metrics in each scenario. Putting thought into specific, measurable goals for each scenario helps zero in on exactly what kind of experience you are aiming for. For instance, when looking for a photo of my daughter in my vast photo archive, is it more important to achieve a sense of magic (that the software somehow knows how to distinguish my daughter's face from my son's), or simply to get the job done quickly (where a simple text search for her name might be close enough—and much easier to build)? What aspects of a solution are going to drive customer delight in this situation? How might you measure them? Later in the project, these metrics can be listed on a scorecard to provide a view of how well you are achieving your goals across all of your scenarios and to help you know when you are "done" and ready to ship.

Scenarios are a popular and effective method for framing software projects, and they have some strong benefits. However, it's important to note that they are not the only possible approach, or even always the best method for every team. Other options include goals and non-goals, requirements, user journeys, outcomes, Agile user stories, and epics, some of which share characteristics with scenarios. Each tool has its pros and cons, but the essential job at this stage is to clearly frame the problem and define what success looks like *before* you jump into building anything. For the purposes of this book, we often refer to scenarios as the most typical framing tool, but you can easily substitute a different approach if that is appropriate for your situation.

Chapter 6, "Framing the problem," discusses framing and measuring against that frame in more detail.

Brainstorm



Now that you have the problem clearly framed, it's time to start considering possible solutions. You want to take the time to explore—to brainstorm and play around with lots of ideas before you make any decisions about which of those ideas you will pursue. At this point, you aren't concerned with finding the single perfect idea; rather, you're generating lots of ideas to give yourself plenty of choices that you can mix and match throughout the iterative process. There are many techniques to help you generate creative, innovative ideas—and it truly is possible to get better with practice.

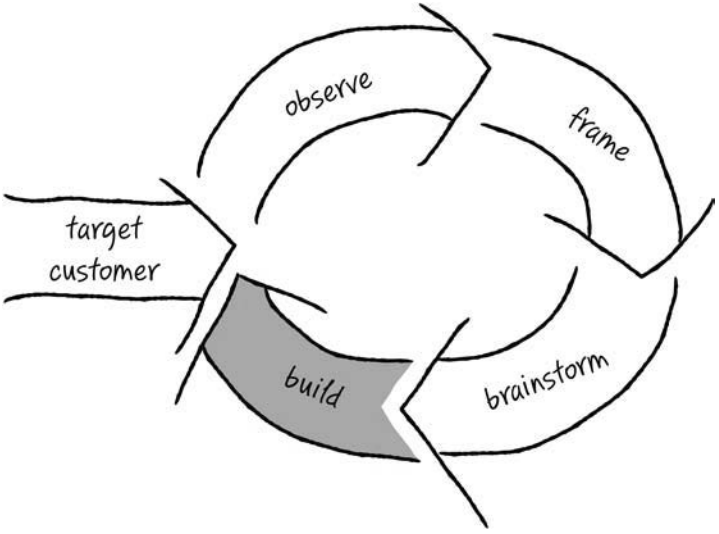
MINDSHIFT

Do the due diligence to generate multiple alternatives before deciding. Often you will find that the best concept emerges as a blend of multiple brainstormed ideas, and you might not have noticed this if you hadn't taken the time to come up with lots of alternatives first. Some companies embed this principle in a rule: you cannot make a decision about *anything* until you have done the due diligence to generate at least X number of alternatives first. The most common number we've heard is five—you need to generate at least five ideas before you are allowed to narrow down the choices and select one. We've heard house rules as high as nine or as few as three. Those are minimums, not maximums—in practice, some teams and companies regularly generate hundreds of ideas as a matter of course, especially in early turns of the iterative cycle. Note that this rule applies not just to deciding how many storyboards to generate or UI layouts to wireframe, but also to how to solve architectural problems or even what to have for lunch. Whatever minimum benchmark you set for yourself, the idea is to get serious about forcing yourself to generate multiple credible alternatives as part of the due diligence of doing engineering and to fully explore the problem space. Do not allow yourself to fall in love with your first good idea.

When you're brainstorming, it is important to think through "complete" solution ideas—that is, ones that solve the entire end-to-end scenario and are not just snapshots of one piece of functionality. It is also very helpful to visualize your ideas, whether in a sketch, a flow chart, or a comic strip. Storyboarding is a very popular technique at this stage because it allows you to visualize a sequence of events that map out a potential end-to-end experience, yet it is lightweight enough that you can sketch dozens of alternatives in an hour or two.

We'll discuss storyboarding as well as other brainstorming and visualization techniques in Chapter 7.

Build

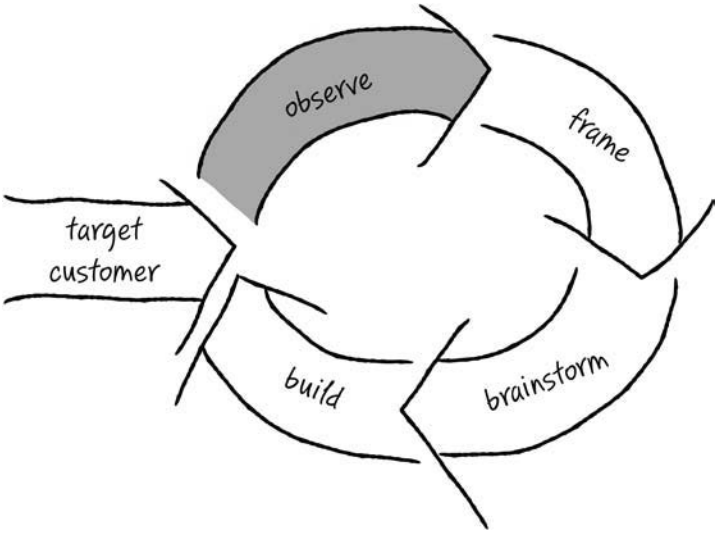


After you explore the range of possible end-to-end solutions, it's time to flesh out a few of those ideas in more detail—not all of them, but the ones that seem to have the most promise from a business, technical, and customer-experience perspective. It's essential to find very cheap, fast, lightweight ways to create prototypes for your first few iterations through the Fast Feedback Cycle—to be as efficient as possible and minimize wasted time on dead ends. You can often do this without writing any code, even when you're working on deeply technical platform components or APIs. The goal of these early prototypes is to get customer feedback on some specific approaches as quickly as possible so that you can make quick course corrections with a minimum of sunk cost.

Your prototypes can be paper drawings, wireframe mockups in PowerPoint, SketchFlow drawings in Expression Blend, an API interface written out on paper, flow charts or block diagrams in Visio, or a skit designed to work out a process workflow or customer-service scenario. The format of the prototype is much less important than how well it facilitates quick feedback from customers about whether you are on the right track. It's important to choose the prototyping technique that works best for you, depending on the type of solution you are working on and the feedback you are looking for. When you finish prototyping, write code in slices to enable continual customer feedback.

We'll discuss prototypes and coding techniques for many different kinds of situations in Chapter 8, "Building prototypes and coding."

Repeat: Observe customers again to get feedback



Now that you have a few prototypes, you start the cycle over again by observing customers using your prototypes. After all, the reason you built those prototypes is to get feedback from customers as quickly as possible so that you can learn and make course corrections while it is still cheap and easy to do so.

Ideally, your first prototypes are little more than simple sketches on paper. You want to learn as early as possible if you are on the right track or where your ideas are falling short. In the extreme, your first prototype for customer feedback could be the scenario you wrote earlier, to be sure that it rings true before you spend any time on solutions for an irrelevant problem.

MINDSHIFT

Get feedback early, really early. Customers are actually much better at dealing with rough prototypes than they're often given credit for. In fact, counterintuitively, showing customers very early prototypes can often result in better-quality feedback because customers subconsciously see that you are still in the early stages of your project, you haven't decided yet, their feedback might actually be heard, and your feelings won't be hurt if they tell you that they don't like your approach. Some engineers tend to resist sharing their thinking until they've thought everything through, but this tendency makes it harder to get the immediate customer feedback needed to power a fast, efficient feedback cycle. *It is important to become comfortable showing early, half-baked ideas so that you have the opportunity to make a course correction before you have too much sunk cost that may end up being wasted effort.*

When you show rough, early prototypes to customers, you learn an amazing number of things. You may find out that your approaches resonate with the customer and that you're on the right track, so you can accelerate your team's work with confidence. You may identify a preference among the alternatives you're considering. Or you may discover that your solution doesn't make sense to customers at all, that you forgot to consider a key constraint, that you addressed only part of their end-to-end problem, or that the overall approach is flawed. It is not uncommon to discover in this first customer touch point that the problem you thought you were solving isn't really a problem—or that you misunderstood the essence of what needed to be solved. Or perhaps what you thought would be a key source of delight for your customers just doesn't resonate with them.

All of these possible outcomes have dramatic impact on the future of the project—and might even cause you to scrap it and focus on a different scenario altogether. Yet, in a typical software project, you wouldn't get this kind of feedback until the code is well on its way, and possibly already working. We've all been in situations where we discover way too late in a product cycle that we made a bad assumption somewhere along the line, that there was a crucial misunderstanding or a critical missing component. The idea here is to ferret out those big oversights much earlier, while it's still cheap and easy to recover. For highest efficiency, optimize your approach to verify that you've got the big picture correct first, before you spend too much time on the details, and certainly before you invest in writing production code. That way, if you find that you need to change direction, you haven't wasted much time or effort. Showing your early, incomplete thinking to customers is a powerful and effective way to catch these big bugs early.

We'll discuss ways to get customer feedback on your prototypes in more detail in Chapter 9, "Observing customers: Getting feedback."

MINDSHIFT

Plan to get it wrong. One of the most important concepts in this book is to plan to get it wrong. Despite plenty of personal experience to the contrary, we all tend to be optimists and believe that this time we will finally get it right on the first try. A more realistic approach is to assume that your first attempt will be partly wrong, you just don't know which part. *The best insurance is to put a feedback loop in place to figure out as soon as possible where the problems are and build time into your project schedule to spend some cycles iterating.*

Think back in your history. Has any product you've ever shipped turned out exactly the way you first imagined it? No, you made plenty of adjustments along the way, sometimes even quite big ones. Those changes were expensive, too, and everyone knows that the later they happen, the more expensive they become to fix (and more work will have been wasted).

Yet we treat those changes as the exception, not the rule. We think that if only we were smarter, we would have come up with the right plan the first time. The problems seem preventable, yet the pattern seems to happen every time. We promise ourselves to do better next time. The trouble is that the world is so complex, and the problems we are solving now are much harder than they used to be—the easy problems were solved a long time ago. You can't predict everything, and even if you could, the landscape is always changing. So stop beating yourself up about it. Instead, plan for the fact that change will happen.

Assume that you will get it wrong the first time, and reserve the right to become smarter as the project progresses.

Keep iterating

You've surely noticed that this approach isn't a linear progression of steps but rather a cycle. In fact, the most important aspect of the Fast Feedback Cycle isn't any single step, but rather the fact that you repeat them. The faster you loop through the cycle, the better the results you are likely to get and the more efficient your team will be. We call each trip around the cycle an *iteration*.

VOCAB

An *iteration* is a single trip around the Fast Feedback Cycle, starting with observing customers, then framing the problem, then exploring many ideas, and finally prototyping or building a few possible solutions to present for customer feedback in the next iteration of the cycle. We say that a team *iterates* as it makes multiple trips around the Fast Feedback Cycle, progressively narrowing in on an optimal solution as it gets customer feedback at the start of every iteration.

Here is a description of how your thinking might progress as you continue iterating after your first trip around the Fast Feedback Cycle:

- **Observe** So, what happens after you show your first prototypes to customers? Well, it all depends on what you learn from the customer feedback you receive. Perhaps customers loved what you showed them. More likely, they liked parts of it, but other parts didn't seem relevant to their needs or just plain didn't make sense.
- **Frame** Now compare the customer feedback against your scenario. This may be a very quick check to confirm that the feedback you received is consistent with the scenario you originally wrote. However, perhaps the feedback suggests that you need to adjust the scenario. Maybe your customers' actual needs or motivations are different from what you initially thought. Maybe their judgment of what a good solution looks like is different, which would suggest that you change the success metrics. Maybe there is a constraint that you didn't realize—for instance, you didn't consider that your customer might not have reliable access to the Internet when using your service on the road. Any of these things might cause you to edit your scenario to include these new needs and insights.
- **Brainstorm** Next, you move on to exploring some revised alternatives, this time with new information to inspire your idea generation in slightly different directions. Perhaps the solution alternatives you originally prototyped didn't quite hit the mark, so you need to try a different approach. Or, perhaps your high-level solutions were working fine with customers, so now your brainstorming is focused primarily on getting to the next level of detail, to bring your rough solutions to a higher level of fidelity. But still, the key behavior is to generate more ideas

than you need so that you have done the due diligence to really explore the problem space before you decide which ideas are worth pursuing.

- **Build** Then, as before, you move on to building or prototyping, but this time you create fewer alternatives and develop them in somewhat more detail than in the previous iteration. You still use the fastest prototyping techniques that are sufficient to get customer feedback. Perhaps after seeing what worked for customers, you can combine ideas from multiple approaches to create an even better prototype. At this stage you should also start exploring which alternatives not only delight customers but are also possible and reasonable from an engineering perspective and support your business plan. You also transition into writing production code.
- **Observe, again** Then you get that second round of prototypes right back in front of customers, just as you did before, and so it continues. But be careful that you don't ask for feedback from just anyone—be sure that you prioritize getting feedback from your target customer.
- **Keep going** The Fast Feedback Cycle can be repeated over and over, and, indeed, you should continue repeating it throughout your project, all the way until the endgame, when you have a final, shippable solution that customers find delightful and that works flawlessly.

As your project progresses, your iterations through the Fast Feedback Cycle will naturally shift their core activities, even though the essence of each stage remains the same. For instance, in early iterations, the framing stage is mostly about deciding which problem to solve and making sure you have described it correctly. As you start iterating, you might realize that your understanding of the problem is incomplete, so you normally update your framing once or twice. In later iterations, this stage becomes more about assessing your solution against that frame to be sure that you are still solving the problem you initially set out to address and assessing how close you are to achieving your metrics.

Similarly, in later iterations, the build stage becomes less about prototyping multiple alternatives and more about writing production code, fixing bugs, and fine-tuning the details. As you shift from showing mocked-up prototypes, to having customers attempt to use working code, to finalizing production code, the techniques you use to gather customer feedback will change as well.

TIP

A rule of thumb for any good-size project is to iterate three times around the Fast Feedback Cycle before you commit to a spec, start writing large quantities of production-quality code, or otherwise finalize the plan. That gives you three opportunities to identify course corrections through customer feedback, even if that feedback happens in a lightweight, informal way. For most teams and most situations, this is the right tradeoff between taking sufficient time to ensure that you're on the right track before you invest too deeply and not getting stuck in analysis-paralysis in an effort to keep making the plan "just a little better." For very large projects, you may find that you need a few more initial iterations to solidify a plan.

After you go through the Fast Feedback Cycle a few times, you should have a concept for an end-to-end solution that you like—one that works from a feasibility perspective and is testing well with customers. Only then is it time to start breaking that solution down into the bits and parts of features and work items that will populate your product backlog, Team Foundation Server, or whatever system you use for tracking engineering work items. But even after you start writing production code, continue iterating: test your latest builds with customers, check against your scenario to be sure you are still solving the customers' core problem, and whenever you find that you have to make a design decision (addressing customer feedback, building the next feature, or fixing a bug), take a couple of minutes to brainstorm some alternatives and perhaps even quickly test them with customers before you pick an approach to move forward with.

MINDSHIFT

Faster is better. The Fast Feedback Cycle is most efficient and effective when you are able to iterate quickly. To make that happen, you need to let up on your inherent perfectionism and let the first couple of trips around the cycle be pretty loose. You don't have to have finished, statistically perfect, triple-checked customer research to start writing a scenario—a well-informed hunch based on talking with a few target customers could be plenty to start with. Nor does your scenario need to be perfect—take your best shot, brainstorm, and prototype against it, and you'll quickly see whether you're on the right track when you show it to customers. That first touch point with customers will give you the information you need to figure out what additional research you should do and where your scenario is not quite accurate, which will help you identify the root of the problem and the optimal solution that much faster.

It's important to note that the rhythm of constant and rapid iteration around the Fast Feedback Cycle should continue throughout the project, even during milestones that are primarily about implementation or stabilization. Iteration is not just for the early stages of a project. Obtaining continual customer feedback on your work in progress is essential for maximizing efficiency and minimizing wasted effort, and it may cause you to rethink the need for a separate implementation or stabilization milestone in the first place.

Looking deeper

An interesting pattern that's embedded in the Fast Feedback Cycle is worth pointing out. As Figure 3-2 shows, the top half of the cycle is all about clearly *understanding* the problem you are trying to solve. The bottom half of the cycle is all about *creating* an actual solution. As the cycle plays out over time, this split ends up being a sinusoidal sort of rhythm—identify a problem, try a solution, see whether it works, adjust based on what you learn, try a different solution, and keep on cycling. It's very powerful to go back and forth between understanding and creating based on that understanding, which then leads to deeper understanding, from which you can create something even better, and on and on. Expert practitioners sometimes call the creating phase *learning by making*, as

compared with *learning by thinking*, and believe that it is a crucial skill to master to get the most out of an iterative approach.

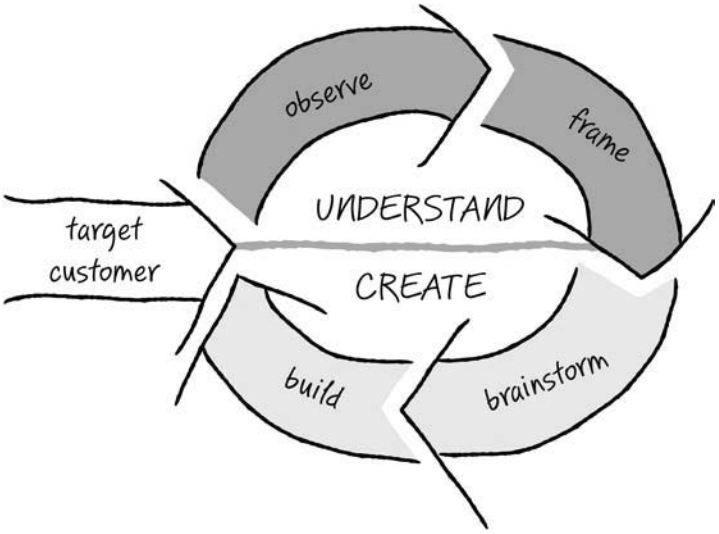


FIGURE 3-2 The top half of the Fast Feedback Cycle focuses on understanding the problem; the bottom half focuses on creating a solution.

Achieving a balanced rhythm across all stages of the cycle is ideal, though different teams will have natural strengths in different parts of the cycle. Predictably, most engineering teams are very strong at building. However, can they build rapid prototypes as well as they write code? And are they equally strong at generating and exploring multiple alternatives? Is the team practiced at identifying insights from customer research? Is the team good at framing a problem to identify the heart of the most important end-to-end experience to solve? What are the strengths of your team?

Haven't I met you somewhere before?

The fundamental concepts of the Fast Feedback Cycle have been around a long, long time. In fact, you can see a very similar cyclical rhythm in many other creative domains:

- Architects who draw and build foam core models to review before they finalize their plans.
- Artists who do pencil sketches before pulling out the paints, and who paint in layers, adding detail and refining as they go.

Or, bringing it to a more technology-oriented realm:

- Agilists who cap each weekly sprint with a demo or touch point to get feedback from their customer or product owner before starting the next sprint.
- Entrepreneurs who use the Lean Startup build-measure-learn cycle to quickly find and validate new business opportunities.

People from diverse backgrounds and disciplines have independently concluded that a fundamentally iterative, cyclical approach is the most effective, efficient way of developing new ideas.

The scientific method

We don't think this is a coincidence—at their core, all of these approaches stem from the scientific method.

While the scientific method is applied with some variation in different domains, it is generally accepted that it entails several key stages. First, you observe some phenomenon and notice something unusual or unexplainable about it. You then identify a specific question you want to answer. Next, you consider various hypotheses about how to answer that question to explain the phenomenon you observed and select a hypothesis to test. Then, you design an experiment and predict what you expect will happen, which outcome would give you the ability to prove or disprove your hypothesis. You then conduct the experiment and collect data. If the data disproves your hypothesis, you repeat the process, first confirming that you are asking the right question, then trying a different hypothesis and experiment, and on again until you land on a hypothesis that actually works to explain the phenomenon you observed. Scientific rigor is achieved by continually challenging the dominant hypothesis through other scientists repeating an experiment in a different lab and by designing new experiments to continue testing the validity of the hypothesis in other situations. The parallels to the Fast Feedback Cycle are plainly obvious, as you can see in Figure 3-3.

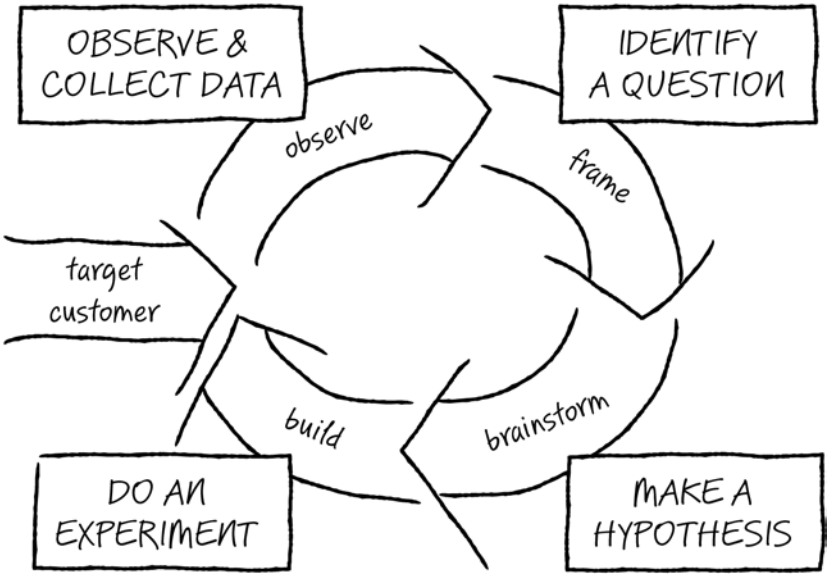


FIGURE 3-3 The Fast Feedback Cycle has much in common with the scientific method.

Historically, many groups of people have approached problem solving in a way that also has roots in the scientific method. Following is a brief survey of the main inspirations for the ideas behind

Scenario-Focused Engineering and the Fast Feedback Cycle, as well as related methodologies that have substantial overlap in methods and mindsets.

User-centered design (UCD)

The most substantial inspiration for Scenario-Focused Engineering comes from the field of user-centered design. Both academics and practitioners have been developing, studying, using, and applying user-centered design techniques for decades. Many universities offer degrees in interaction design, user research, and numerous related specialties.

The core ideas behind user-centered design are central to this book: develop empathy with your customers, use observation to discover unarticulated needs, use storytelling to frame the problems to be solved, brainstorm and visualize alternative solutions, test prototypes with customers to get feedback, iterate repeatedly at increasing levels of fidelity in response to customer need, and mindfully narrow the funnel of solutions at each iteration to ensure that the best amalgamated solution is delivered. The fundamentally experimental, cyclical nature of a user-centered design approach is a close analog of the scientific method and is embodied in the Fast Feedback Cycle.

A mainstay of user-centered design is a form of logic called *abductive reasoning*.[3] Unlike deductive or inductive logic, abduction is a form of logical inference that takes a set of data and creates a hypothesis that could possibly explain that data. When designing solutions, abduction is a logical leap of mind that suggests a reasonable design that could solve an attendant problem. Logically, you can't know for sure that this solution will work, but it might. Abduction creates a hypothesis that you can test, measure, validate, or disprove—just as a scientific hypothesis is tested through the scientific method.

In the past few years, the field of user-centered design has been broadened and is often referred to as *design thinking*, which is a recasting of the same core ideology and methods, recognizing its applicability to a much wider class of problems than just user-interface design. We concur that as a variant of the scientific method, these approaches are indeed very broadly applicable. They can be applied not just to designing products and services, but also to developing new business strategies, designing optimal organizations and systems, and even solving seemingly impossible world-scale problems, such as creating viable ways for people to get out of poverty and improving community access to clean water.

Agile

Agile was born from the software developer community in response to the need to build higher-quality software with less wasted effort in an environment in which precise customer requirements change over time and are hard to articulate upfront. Agile was invented independently by software engineers, but fascinatingly, it aimed to solve a lot of the same root problems that user-centered design aimed to tackle.

If you're working on an Agile team, a lot of this chapter probably feels familiar. You already loop quickly in sprints, likely somewhere from one to four weeks in length. It's easy to squint and see how one loop around the Fast Feedback Cycle could map to an Agile sprint, complete with a customer

touch point at the end to get direct customer feedback on the increment you just delivered. It's likely that some of the same activities we discussed already occur during the course of your sprints.

Scenario-Focused Engineering differs from Agile in two main areas, which we believe help fill in some missing gaps. First, we believe that it's not necessary for every sprint to deliver shippable code. In early sprints, it's often more efficient to get customer feedback on a sketch, mockup, or prototype before investing in production code. However, we completely agree that getting customer feedback at the end of every sprint is absolutely essential, whether that sprint built prototypes or production code.

Second, we believe that the product owner is actually a mythical creature. We challenge the idea that any one person on the team, no matter how senior, no matter how often they talk with customers, can accurately channel exactly what real customers need and desire and provide accurate "customer" feedback at the end of a sprint. Agile oversimplified the whole business of understanding customers by giving that job to one person—the product owner. Anything beyond the most basic customer needs are too complex for that approach to work reliably in today's market. We hope you'll find that the ideas in this book give you practical ways to break out of that mold and actually talk to real customers.

Interestingly, most Agilists have concluded that shorter sprints work better than longer ones: sprints of one to two weeks are better than sprints of three to four weeks. We have come to a similar conclusion; you should probably aim to finish a single cycle within two weeks. Left to their own devices, most teams will spend too long on an iteration. Time boxing is key to keep people moving, and Agile sprints are a really natural way to time box on a software project.

Lean Startup

The ideas of Lean Startup were made popular by Eric Ries's book of the same name and are rooted in a combination of Steve Blank's "Four Steps to the Epiphany" approaches to entrepreneurship and the continuing evolution of lean approaches that started with "lean manufacturing" at Toyota in the 1990s.[4] The lean approach believes in finding ways to achieve equivalent value but with less work. Lean Startup takes this core belief and applies it from a business-minded entrepreneur's frame of mind to identifying and incubating successful new business ventures. However, the ideas in it are just as relevant to developing new services, products, and offerings in established companies as they are for an entrepreneurial startup.

There are particularly strong parallels between the Fast Feedback Cycle, the scientific method, and Lean Startup techniques. The build-measure-learn cycle is really just a simpler statement of the essence of the scientific method: experiment, collect data, create hypothesis. Lean Startup's focus on "innovation-accounting" measurement techniques emphasizes the importance of collecting data rigorously to prove or disprove your hypothesis, and to not allow "vanity metrics" or other psychological biases to get in the way of objectively assessing progress against your actual business results. The idea of a faked-up home page for your new offering is another form of a rapid, low-cost prototype intended to get customer feedback as quickly as possible to verify that you are on the right track with a minimum of sunk cost. As Ries would say, your goal is to reduce your "mean time to learning."

Similarly, a "minimum viable product" is a powerful concept to help keep a team focused, usually on solving a single scenario and seeing how it does in the market before branching out.

Mix and match your toolbox

At the core, these approaches are inherently compatible because they are fundamentally based on the rhythm of the scientific method. Therefore, it's ridiculous to say, "We have to decide whether to use Scenario-Focused Engineering or Agile for this project." Or "Is it better to go with user-centered design or Lean Startup?" It absolutely does not need to be an either/or proposition.

Rather, the questions to ask are "Which Agile techniques would be helpful for this project?" and "Which Scenario-Focused Engineering techniques would apply?" and "Which Lean Startup techniques would work?" Each methodology provides a set of techniques that enrich the toolbox; these techniques are born from different perspectives and emphasize different sets of expertise, but they are all fundamentally compatible. Use the tools and techniques that work for your situation and your team. When you consider them in the context of the Fast Feedback Cycle, you will find that they fit together nicely.

As we detail each stage in the Fast Feedback Cycle in the following chapters, you will see that you can choose from dozens of different specific techniques within each stage. In most cases, we encourage you to pick one or perhaps only a small handful of techniques to use at any given time—certainly don't feel as though you have to do it all, or even think that doing it all would be ideal. Actually, the idea is to select the fewest number of techniques and activities at each stage that give you just enough information to move forward to the next stage as quickly as possible. Remember that fast iterations are the key!

We have coached many different teams as they adopt an approach based on Scenario-Focused Engineering, often with a mix of Agile or Lean Startup (and sometimes both) mixed in. Our experience is that this is not a cookie-cutter process, and there is no single best recipe that works for everyone. Each team is responsible for building its toolbox to complement the scale of its project, its timeline, the skills already on the team, and all the other myriad factors that make each team unique. Each team's toolbox is different. However, we've found that a few tools are used more often than others, and these are the ones we elaborate on throughout the next chapters in sections that we call "Deep Dive."

If you aren't sure where to start, rest assured that experts are available for nearly every topic discussed in this book. We'll highlight the places where getting expert help is a particularly good idea.

MINDSHIFT

You're probably noticing by now that there is some pretty heavy stuff embedded in Scenario-Focused Engineering. On the surface, the Fast Feedback Cycle looks easy and straightforward. But actually implementing it in an engineering team is far from simple, and there are many questions to answer. We've worked with a lot of teams, and there are common issues, questions, and tradeoffs that come up with nearly every one.

How do you schedule for iteration when you can't predict how many iterations you need to get to a great solution? How can you write scenarios when you're not sure what your customer really needs? Who is going to do all of that customer research anyway? Should every developer really go visit customers instead of writing code, or is reading a trip report sufficient? Who exactly is our target customer? How will we get access to customers to test early prototypes quickly without a lot of overhead? Realistically, can we actually plan out an entire engineering release without a feature list and focus on a small number of end-to-end scenarios instead? How can we keep focus on scenarios all the way to the endgame—what project reviews, status-tracking systems, or scorecards will we need to keep us honest? Will our leadership actually support this approach? Does our office space, performance-review process, and team culture support the kind of open collaboration we need for this approach to work?

Probably the most significant aspect beyond any individual question is the fact that you need the entire engineering team to use this approach for it to actually work. Indeed, this is one of our biggest insights from starting the Scenario-Focused Engineering initiative at Microsoft, and it has been reinforced over and over again as we've worked with countless teams to adopt these ideas. There are noteworthy changes in schedule, process, and tools, as well as significant shifts in mindset, leadership, and team culture that need to happen to make the Fast Feedback Cycle a reality, especially on a large scale.

If you are one developer on a larger team reading this book, what do you do? You can't change the whole organization, can you? Take heart, the good news is that you don't have to do the entire Fast Feedback Cycle to get some nice benefits—even just brainstorming more intentionally, or taking the time to write a scenario before you jump headlong into solutions, will give you plenty of incremental gains. You don't have to do everything; you can still get a lot of benefit by making use of just a few of these tools and techniques. Furthermore, as excellent as this book may be—we are sad to say that the chances of you reading this book and becoming an expert practitioner overnight are pretty slim. It takes most teams years to truly make this shift, so be patient with yourself. Appendix A includes the SFE capability roadmap, which is a checklist of skills and behaviors that can help you plan your team's progress over time and prioritize which skills are most important for your team to develop and improve on.

We have seen many teams get started very simply. A single champion on the team identifies a couple of practices that make sense to try in an incremental way. That champion recruits a few buddies on the team to help make it happen on one small project. After those first attempts show some success, the larger team becomes more receptive, the leaders agree to train the rest of the team, and the practice continues to develop from there. It's worth noting that even teams that have tried to "do it all" found that there was a practical limit to how much change their organization could absorb at one time. We'll continue discussing the team and cultural impacts of adopting Scenario-Focused Engineering throughout the book and in detail in Chapter 12, "Lessons learned."

Summary

Good product design follows a predictable, age-old pattern that can be learned and replicated. This pattern is encapsulated in the rhythm of the Fast Feedback Cycle, which is an application of the scientific method to the business of designing and delivering software products, services, or devices. By observing customers, framing their key scenarios, exploring multiple ideas, prototyping rapidly, and judging your progress against customer feedback and predetermined metrics, you can quickly discover whether you're on the right track to delight your target customers, with an absolute minimum of sunk cost.

Notes

1. Interestingly, better ergonomics wasn't something that customers were asking for per se, as people didn't seem to think it was possible to make a mouse more ergonomic. This is a great example of an unarticulated customer need.

2. The ergonomic mouse met all its sales targets, despite being released just at the start of the recession and despite being a high-end product for a niche market. But it didn't get universally great reviews—in fact, the reviews were somewhat polarized. There were many customers who absolutely loved this mouse. Then there were others who complained that it didn't work at all for lefties or that the buttons didn't work well for smaller hands or for very large hands. The product team made a conscious choice to focus on right-handed people of average size knowing that it couldn't do its ergonomic goals justice for a broader group, which was a key constraint the team figured into its business plan and sales targets. But for the target customer, this mouse was a dream come true and had a loyal following.

3. Abductive logic was originally coined by Charles Sanders Peirce (http://plato.stanford.edu/entries/peirce/) in the early 1900s as he described the interplay between abductive logic (making hypotheses), deductive logic, and inductive logic in the scientific method. More recently, Roger Martin discussed the role of abduction in design thinking in his books *Opposable Mind: Winning Through Integrative Thinking* (Harvard Business Review Press, 2009) and *The Design of Business: Why Design Thinking Is the Next Competitive Advantage* (Harvard Business Review Press, 2009).

4. Eric Reis, *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses* (New York: Crown Business, 2011); Steve Blank, *The Four Steps to the Epiphany,* 2nd ed. (K&S Ranch, 2013).

This page intentionally left blank

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

This page intentionally left blank