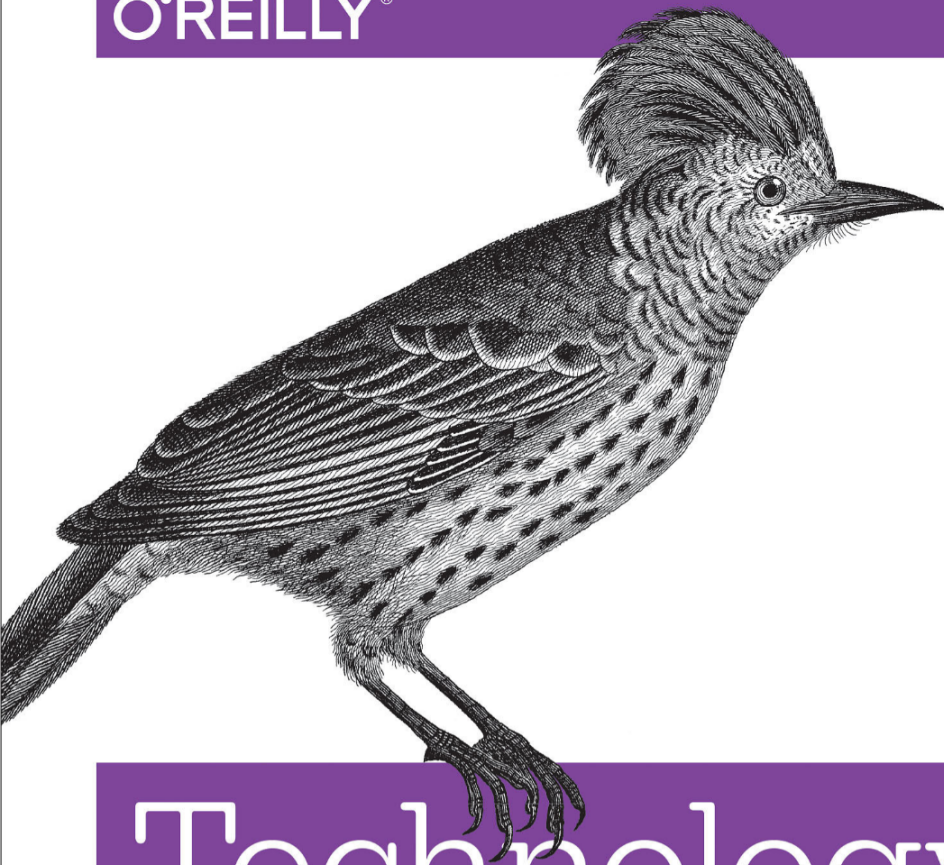


O'REILLY®



# Technology Strategy Patterns

---

ARCHITECTURE AS STRATEGY

Eben Hewitt

---

# Technology Strategy Patterns

*Architecture as Strategy*

*Eben Hewitt*

Beijing • Boston • Farnham • Sebastopol • Tokyo

**O'REILLY®**

## Technology Strategy Patterns

by Eben Hewitt

Copyright © 2019 Eben Hewitt. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://oreilly.com/safari>). For more information, contact our corporate/institutional sales department: 800-998-9938 or [corporate@oreilly.com](mailto:corporate@oreilly.com).

**Acquisitions Editors:** Brian Foster, Mary Treseler

**Development Editor:** Alicia Young

**Production Editor:** Nan Barber

**Copyeditor:** Rachel Monaghan

**Proofreader:** Sharon Wilkey

**Indexer:** Ellen Troutman-Zaig

**Interior Designer:** David Futato

**Cover Designer:** Karen Montgomery

**Illustrator:** Rebecca Demarest

October 2018: First Edition

### Revision History for the First Edition

2018-10-02: First Release

See <http://oreilly.com/catalog/errata.csp?isbn=9781492040873> for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Technology Strategy Patterns*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

The views expressed in this work are those of the author and do not represent the publisher's views. While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-492-04087-3

[LSI]

---

# Table of Contents

<b>Preface</b> .....	<b>vii</b>
<b>Introduction</b> .....	<b>xv</b>
<hr/>	
<b>Part I. Context: Architecture and Strategy</b>	
The Origins of Patterns	1
Applying the Patterns	2
<b>1. Architect and Strategist</b> .....	<b>3</b>
Business Strategies	3
The Architect's Role	7
The Strategist's Role	15
Summary	21
<hr/>	
<b>Part II. Creating the Strategy</b>	
A Logical Architecture of the Creation Patterns	23
<b>2. Analysis</b> .....	<b>27</b>
MECE	29
Logic Tree	37
Hypothesis	41
Strategic Analysis as Machine Learning	66
Summary	67

<b>3. World Context.....</b>	<b>69</b>
PESTEL	70
Scenario Planning	77
Futures Funnel	80
Backcasting	83
Summary	85
<b>4. Industry Context.....</b>	<b>87</b>
SWOT	87
Porter's Five Forces	89
Ansoff Growth Matrix	95
Summary	97
<b>5. Corporate Context.....</b>	<b>99</b>
Stakeholder Alignment	99
RACI	108
Life Cycle Stage	111
Value Chain	116
Growth-Share Matrix	124
Core/Innovation Wave	126
Investment Map	130
Summary	132
<b>6. Department Context.....</b>	<b>133</b>
Principles, Practices, Tools	133
Application Portfolio Management	146
Summary	158

---

## Part III. Communicating the Strategy

<b>7. Approach Patterns.....</b>	<b>161</b>
30-Second Answer	161
Rented Brain	163
Ars Rhetorica	167
Fait Accompli	174
Dramatic Structure	179
Deconstruction	185
Scalable Business Machines	194
Summary	211

<b>8. Templates.....</b>	<b>213</b>
One-Slider	214
Use Case Map	217
Directional Costing	218
Priority Map	226
Technology Radar	227
Build/Buy/Partner	229
Due Diligence	232
Architecture Definition	235
Summary	250
<b>9. Decks.....</b>	<b>253</b>
Ghost Deck	253
Ask Deck	256
Strategy Deck	259
Roadmap	260
Tactical Plan	261
<b>10. Bringing It All Together.....</b>	<b>265</b>
Patterns Map	265
Conclusion	267
<b>A. Recommended Reading.....</b>	<b>269</b>
<b>Index.....</b>	<b>271</b>



---

# Preface

## Welcome

Thank you for picking up *Technology Strategy Patterns*.

This book came out of a paper I gave at the O’Reilly Software Architecture Conference in New York City in the spring of 2018, called “The Architect as Strategist.” I’m grateful for the many conversations it sparked. At the conference, a number of the architects in attendance asked if it could become a book. And so it is.

## Intended Audience

This book is for anyone in information technology who wants to do more strategic, relevant, important work for their organizations. Therefore, there is no code in the book, and nothing too technical. People who will get the most out of it include:

- Architects
- Principal developers or tech leads who wish to become architects
- Technology managers in engineering, testing, and analysis, whether on the product development side or the IT back office
- Product managers
- Project and portfolio managers
- Business consultants
- Technology executives



- Strategy analysts and managers
- Anyone interested in strategy, architecture, and leadership

Whether you are a senior developer, enterprise architect, or CTO, or have never read a line of code in your life, I know you'll find something useful, and feel welcome and at home here.

## Purpose of the Book

The book has two aims. The first is to help architects, product managers, and executives at technology companies or in technology organizations who are charged with producing technology strategies. This stuff works across industries. My hope is that with these practical tools and guidance, your strategies will be deeper, stronger, and clearer, and you'll get approval, support, and funding to make your ideas a reality. The primary assumption of the book is that you're in technology management of some kind, or want to be, and want to think more holistically and incisively about your technical roadmaps. The second aim is to help you in your career. I suppose an alternate, but less becoming, title for this book could be *How to Become the CTO*.

If you're familiar with patterns-oriented books, such as the Gang of Four classic *Design Patterns* (Addison-Wesley), this book takes inspiration from them without adhering too tightly to the template they typically employ. One of my favorite books, and one that changed how I think about software and the evolution of ideas, was *A Pattern Language* by Christopher Alexander (Oxford University Press). I have devised and refined this *bricolage* of ideas over several years from this and many sources. This book is, in a sense, just a written record of how I've approached this aspect of my work, as much an intellectual memoir as anything else.

While I've written books before on Cassandra, the Java programming language, software development, architecture, SOA, and web development, *Technology Strategy Patterns* is my first real book in nearly a decade. That's on purpose. I've been developing these ideas for the better part of that decade in my work conceiving and executing strategy as CTO, CIO, and Chief Architect at global tech companies. It represents a synthetic fabric of three areas:

- The first is a set of frameworks borrowed from the world of business strategy consulting as it is conceived in McKinsey,

Bain, BCG, and Harvard Business School. We technologists are often told that if we want to be heard, be understood, and get funding, we must “speak in the language of the business”—without quite being told what that is or how to do it. This book serves as the translation—the Rosetta Stone, if you will—for the language of business executives to teach you what they know to strengthen your work and help it succeed.

- Second, I borrow from the world of philosophy, having studied and fallen in love with it in graduate school, and finding its rigors and explosive power very helpful in my 20-year career in tech.
- Finally, there are many perhaps idiomatic tools and frameworks that I developed myself while running large teams of engineers, helping grow businesses, instituting organizational and cultural changes, and designing and implementing globally scalable, mission-critical, distributed software systems running thousands of transactions per second. Together, they form an array of lenses that you can variously employ over time in different contexts as needed. They’ll help you define, create, elaborate, and refine your architecture goals and plans, and communicate them in rhetorically powerful ways to an audience of executives who must approve them as well as the teams who must implement them.

I recommend that you read the book front to back. As the Mad Hatter says, “Begin at the beginning, and when you get to the end, stop.” The ideas build on each other, refer to each other, and are carefully organized in a logical architecture of their own to reveal to you the beautiful world of strategy one peek at a time. After you’re done, keep the book handy to refer back to later as needed. You won’t make a new strategy every day, but I think you’ll see how many of the techniques can be woven into your daily work.

The tools in this book are proven. They work. I’ve used these techniques for years, in many contexts with many different leaders in many different organizations. Employing these techniques has repeatedly helped me win technology strategy funding for \$1M, \$10M, \$30M, \$50M, \$75M, and more. If you employ these tools, your ideas will be sharper, your plans more accurate, relevant, empathetic, and fruitful. Executives will approve and fund your work, and

your teams, your company, your customers, and your partners will benefit.

I truly hope that you find this book useful and inspiring for years to come, and that it serves you. It was written with affection and care. May it strengthen and deepen your work, and help you, your company, and your customers succeed.

## Conventions Used in This Book

The following typographical conventions are used in this book:

### *Italic*

Indicates new terms, URLs, email addresses, filenames, and file extensions.

### Constant width

Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

### Constant width bold

Shows commands or other text that should be typed literally by the user.

### *Constant width italic*

Shows text that should be replaced with user-supplied values or by values determined by context.



This element signifies a tip or suggestion.



This element signifies a general note.



This element indicates a warning or caution.

## Using Code Examples


Supplemental material (code examples, exercises, etc.) is available for download at <https://www.aletheastudio.com>.

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: “*Technology Strategy Patterns* by Eben Hewitt (O'Reilly). Copyright 2019 Eben Hewitt, 978-1-492-04087-3.”

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at [permissions@oreilly.com](mailto:permissions@oreilly.com).

## O'Reilly Safari

 **Safari**® *Safari* (formerly Safari Books Online) is a membership-based training and reference platform for enterprise, government, educators, and individuals.

Members have access to thousands of books, training videos, Learning Paths, interactive tutorials, and curated playlists from over 250 publishers, including O'Reilly Media, Harvard Business Review, Prentice Hall Professional, Addison-Wesley Professional, Microsoft

Press, Sams, Que, Peachpit Press, Adobe, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, and Course Technology, among others.

For more information, please visit <http://oreilly.com/safari>.

## How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.  
1005 Gravenstein Highway North  
Sebastopol, CA 95472  
800-998-9938 (in the United States or Canada)  
707-829-0515 (international or local)  
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at <http://www.oreilly.com/catalog/0636920175155>.

To comment or ask technical questions about this book, send email to [bookquestions@oreilly.com](mailto:bookquestions@oreilly.com).

For more information about our books, courses, conferences, and news, see our website at <http://www.oreilly.com>.

Find us on Facebook: <http://facebook.com/oreilly>

Follow us on Twitter: <http://twitter.com/oreillymedia>

Watch us on YouTube: <http://www.youtube.com/oreillymedia>

## Acknowledgments

I would like to thank my friends and colleagues at Sabre who shared important insights and supported this work.

First, to our wonderful enterprise architecture team of Andrea Baylor, Holt Hopkins, Tom Murray, Jerry Rossi, and Andy Zecha. You're such a joy to work with every day. I'm grateful to get to hang out

with such knowledgeable, sharp, funny, dedicated, passionate, good people.

President Clinton Anderson, previously of Bain & Company, and Sabre Hospitality VP of Strategy Balaji Krishnamurthy, previously of McKinsey, both inspire me, are such fun to work with, and teach me every day. I'm so grateful to work with you and know you. Thank you for sharing your thoughts and methods, and in so doing, helping inspire this book. You both get stickers.

A special thank you to Justin Ricketts for his support on this project and for being so terrific to work with. I am grateful to my colleagues across Sabre, especially Tom Winrow.

Thank you to Brian Mericle, enterprise architect, and my longtime friend and colleague, for his valuable edits to this book.

Thank you to my old friends and mentors Todd Davis, Deryl Heitman, Steve Miller, and Ted Taylor for the many opportunities you afforded me to grow and learn.

Thank you to Mike Loukides, my longtime friend and editor at O'Reilly, who always pushes me to see farther. I'm grateful for your help in developing this book and these ideas. Thank you for our sprawling, fabulous conversations. Thank you too to Brian Foster for welcoming this book at the Software Architecture Conference and your guidance of this work. Thank you to my development editor, Alicia Young, whose diligence and care improved this work. Thank you to all my friends at the wonderful O'Reilly Media, including Mary Treseler, Nan Barber, Rachel Monaghan, and Sharon Wilkey for your terrific work that improved this book. I am grateful for what you do in the world to help spread the knowledge of innovators. What a beautiful company you created, Tim; may it ever flourish.

I am most indebted to my wife, Professor of Philosophy Alison Brown, for her wisdom, better ideas, revisions, suggestions, summations, expansions, explanations, expiations, encouragement, tenacity, and love. You are the *sine qua non*, as ever.



---

# Introduction

## This Is Water

My favorite joke is told by philosopher and author David Foster Wallace in [his address to the graduating class of Kenyon College](#) in 2005. It goes like this: One morning two young fish are swimming in the ocean. They come across an older fish who waves happily and calls out to them, “Morning, friends! How’s the water?” They nod in acknowledgment and swim on. Once they’re out of sight, one turns to the other and asks, “What the hell is water?”

With this joke, Wallace reminds us that the most obvious, important realities are often the ones hardest to see, that we can lock ourselves in mental models so complete that we don’t even know we’re imprisoned by them.

As technologists, we can be perhaps particularly susceptible to this. Our work is engaging and requires a watchmaker’s attention to detail. Yet, as technologists, we are businesspeople. A hammer doesn’t exist to be a hammer. It’s a tool to construct something else. Technology is one tool with which businesses are constructed, rise, and fall. We operate in wide spheres of ever-farther-reaching impact on the world around us. In a sense, this book is about constructing a new mental model within this water of business.

## Discovering Strategy

The roles that are ultimately valued at an organization tend to be the people who do what the boss did. If the boss used to be a salesperson or deal-maker, that’s who she’ll recognize, side with, empathize with, reward, understand, and listen to most. If you want your voice to be



heard, you must make a concerted effort to empathize with people, and employ the tools, techniques, and language that they respond to.

At one point in my career I was running the enterprise architecture department of a large corporation. My manager, the CTO, asked me to help him estimate a large project. He wanted me to go off and determine the “incrementals.” I didn’t know what he meant. But in my stupidity, I didn’t want to *look* stupid, so I didn’t ask him, thereby enthroning myself as truly stupid. So I went off and tried to figure it out myself and came back to him three or four times with something different than he needed. I was a pretty good technologist, but I didn’t sufficiently understand the language of business. And therein lies the problem. It’s hard for people to know what they mean themselves, much less express it to others in a way that achieves their aims. After all, that’s the secret to happiness in life: figuring out what you want, and learning how to ask for it.

As I have progressed in my career from developer to architect to CTO and CIO and Chief Architect, I have been asked to create a technology strategy many times. Concluding that no one asks the not-as-clever people to craft strategies for stuff that doesn’t matter, I was always delighted at the prospect. It felt like an honor, sounded really cool, and seemed important and big, like I had been asked to help make decisions about how to guide the organization. So I was over the moon for a moment. And then suddenly scared. Because I realized for all the times I’d heard people say the word as if they knew what they were saying, I had never seen anything that I thought looked like a strategy. My concern grew as I realized many of my (accomplished and perfectly reasonable) bosses hadn’t either. They didn’t know exactly what they were asking me to do, or what the result should look like. Perhaps the strategists were the only people left in the world with a higher room in the Ivory Tower than architects and academics. Eventually we all bumbled our way through it and got to something good enough. But in some cases this process took a year and wasn’t always optimal.

Yet I was intrigued, in part because it wasn’t lost on me that the clever people, and the people running the organization (only occasionally the same thing), were keenly interested in strategy.

While trying to discover what a good strategy should look like, I grew more concerned at being able to construct this seemingly critical but elusive and mystical document. Companies do not tend to

publish their strategies externally since they contain revealing secrets about their plans and fears. So it is hard to find any recent, good, complete, relevant examples. I therefore took it upon myself to go to the source: strategy consultants. They are devoted to publishing their work and excitedly talking about it nonstop to anyone.

But once you've devised a strategy, it languishes on the shelf if you can't make people excited to hear it, understand it, care about it, approve it, and execute it. Any technology strategy is, in a sense, a request to spend millions of dollars of someone else's money. If you think of your work as a technology strategist in this way, you'll do it differently. By which I mean better.

I have known many smart people, wonderful technologists, who do not get their ideas heard by upper management. They state what the problems are and where the problems are going to be, write that up, and put it on the wiki—and nothing changes. Once it's too late and the platform is burning, those same architects get called in to rescue the situation. While people love to say, "I told you so," no one likes to hear it. These well-intentioned souls may have had the best recommendations, but it never mattered. These folks can become alienated, feeling misunderstood and unappreciated. And the business loses out on their great ideas. This is precisely what I don't want to see happen to you, and the reason I wrote this book.

## Driving Strategy with Patterns

This book employs, albeit loosely, a suggestion of patterns that is likely familiar to you from the realm of software design patterns such as Decorator, Factory, Visitor, and Pub/Sub. They're used as shorthand for known, proven solutions, to provide an easy way for us to communicate to each other. I chose patterns to represent the ideas in this book because of that familiarity, and because that structure makes it easy for you to look up these ideas for years to come. To aid in this, they're divided into logical concept architectures.

### *Analysis*

First we explore foundational and general tools for critical thinking that will underpin the other patterns in the book.

### *Creation*

These are the patterns that help you directly create your tech strategy. If you implement all of these patterns, you'll have a

comprehensive, compelling annual tech strategy. But you don't need to always implement all of them. You can also pick and choose individual patterns to take a strategic approach to more local, specific project work.

### *Communication*

These patterns help you to organize the components of your strategy in a way that your colleagues and executives can understand, get excited about, and support.

I'm sorry to repeat an old saw, but it's true: increasingly, it is impossible to distinguish between business and technology. But that distinction is still more powerful than it deserves to be, given typical organizational structures and the resistance to change, and an uncertainty about how to do so. I hope that in part this book will help you, your colleagues, and your organization to embrace this cross-pollination. I hypothesize that in the future, people who can learn quickly as synthetic interdisciplinarians will be highly effective, and highly prized, because maintaining that distinction is increasingly a barrier to progress, creativity, and innovation.

This book, I hope, gives technologists, strategists, product managers, executives, technology managers, and the architects who frequently mediate these worlds all a shared language. In this, may you be more fruitful.

---

# Context: Architecture and Strategy

*All models are wrong; some models are useful.*

—Statistician George Box

## The Origins of Patterns

Christopher Alexander was a professor at the University of California, Berkeley. With a group of graduate students in the mid 1970s, he set out to catalog common practices he saw throughout architecture. He noted that many problems in architecture are inveterate, and that recording a set of optimal, or at least frequently employed, solutions to these problems would help elevate architecture as a field and expedite the work of architects. He called these common solutions “patterns,” and his most excellent book, *A Pattern Language*, catalogs dozens of them.

Inspired by Alexander’s work in the architecture of houses, buildings, and city planning, the Gang of Four applied the idea of patterns to software in their book *Design Patterns*. Since then, many books have employed patterns in a variety of technological domains, and the present work expands on this idea, taking repeated solutions found in the work of business strategists and illustrating how we can apply them to better our work as technologists.

The use of patterns as a structuring mechanism here is intended to make the book easy to use later as a reference after you've read it.

## Applying the Patterns

There are five basic steps to follow in formulating your strategic technology analysis. Here is a simplified outline:

1. Establish context
  - a. Analyze the trends happening in the world outside.
  - b. Analyze the forces at work across your industry, your organization, and your department.
  - c. Gain a view on your stakeholders.
2. Understand your competition, the market, and the technology landscape.
3. Identify strategic options in your products, services, and technology roadmap.
4. Evaluate those options.
5. Make a compelling recommendation with a coherent, cohesive, comprehensive strategy to gain approval and resources to execute your plans.

With this process in mind, let's turn our attention to how to view your technology work through the lens of architecture and strategy, so we have a shared understanding and vocabulary.

---

# Architect and Strategist

This chapter provides an overview of three different, somewhat traditional business strategy examples from different industries. We'll then look at the role of the architect and the role of the strategist in modern business, to see how strategically minded technologists can be a catalyst for real, meaningful change in their organizations.

## Business Strategies

Business strategies reveal how companies allocate resources toward a certain aim. Let's take three examples: Michelin, a tire-company-turned-dining giant; Oracle, a dominant name in software; and Xerox and Canon, companies whose strategies set them on very different paths in the copy industry. Our brief look at these strategies will provide you with context for the concept of "strategy" and illustrate the business implications different strategies can have.

## Marketing at Michelin

The Michelin Guide has been in circulation for nearly 120 years. It is known across the globe as the gold standard for fine dining restaurant ratings and reviews. The world's top chefs work year-round in pursuit of the coveted Michelin star, because being awarded one means that your restaurant is worth a detour, worth making a special trip just to eat there. And thousands of diners trust the guide as a well-known authority on the best restaurants. Such excitement is created in France each year upon its publication that the media frenzy it ignites has been compared to that for the Academy Awards.

But Michelin is a *tire* company. How in the world, and why, did a tire company come to hand out the highest honors in fine dining?

In 1900, there were only a few thousand cars in France. Cars were new, they were relatively expensive, and the culture had not yet shifted toward the idea that everyone needed to own a car. For tire manufacturer Michelin, that presented a problem. How could it sell more tires and thrive as a company when there were so few cars?

There are only two ways to create more demand for its product: sell more cars to outfit with tires, or find a way to make people who already have cars drive more so their tires would need to be replaced sooner. The company created the Michelin Guide and gave it away for free. In doing so, Michelin got its name out across France, then Europe, then the world as an excellent advertisement, and positioned itself as an approachable, authoritative company. It created inspiration for drivers and a reason for more people to have cars, and sold more replacement products as a result. The company also made money on the guide once it started charging for it.

This was an innovative, counterintuitive, winning business strategy that worked well for decades.

The guides, known affectionately as the “red books,” grew the value of the brand overall and seem to be a real asset. Yet today, published on paper and sold in bookstores, the guides lose Michelin €19 million per year. These days, with the ubiquity of cars, and the joys of the open road firmly ensconced in the popular imagination, the guides don’t act as powerfully in their original capacity. Yet they’re still obviously important. They became disconnected from the idea of getting people to drive more, and started to have to run as their own business. Perhaps a new strategy better supported by, and better integrated with, technology could help make it profitable again.

## Acquisition and Integration at Oracle

In 2007, Oracle Corporation determined a business strategy with a simple principle: either make its software number one or number two in every product category, or buy the market leader. In other words, if you can’t beat them, buy them. Between 2008 and 2013, Oracle bought nearly 60 companies—a rate of almost one per month. Oracle spent \$45 billion acquiring companies between 2004 and 2014.

When Thomas Kurian assumed leadership of the product teams for Oracle Fusion Middleware in 2008, his technology strategy was made clear to everyone at Oracle and to its customers: all products would use Oracle's middleware stack and must be modified to interoperate with it.

This technology strategy has turned out to be a mixed bag. On the one hand, it's a terrific example of how a technology architecture decision was made to directly support the business strategy of aggressive acquisitions, and that's a strong lesson to learn.

On the other hand, Oracle spent considerable time over many years on refactoring and redoing the internals of many products to comply with this architecture. That time was not spent on innovation or features for customers. In that time, Oracle entirely missed the critical revolutions in the cloud and machine learning, putting it years behind competitors in those crucial areas. More than 10 years later, the technology strategy and architecture within products remains unchanged.



### **Published Reference Architecture**

A few years ago, Oracle published its set of technology strategies, reference architectures, and practitioner guides in a fairly comprehensive [website](#). This is an excellent example of working to help educate your community on how to best take advantage of your strategy once you've published it.

## **Differentiation at Xerox and Canon**

In 1968, Xerox introduced the 914 copying machine, which was capable of copying at what then was the astonishing rate of 120 copies per minute. With this product, it became the world's fastest company to grow to a billion dollars.

By the early 1970s, Xerox had a 95% market share in the global copier market. The large Xerox machines sold to large corporations with high-volume copy needs. The price: a whopping \$80,000 to \$129,000 each.



Xerox had a sophisticated and sizable sales force, all armed with deep product knowledge. Its mission was to build close, long-term customer relationships with all the Fortune 500.

Reliability was paramount: a stop in the copiers could mean a stop of the customer's business. Because of the centrality of the copiers in the business, Xerox built sturdy machines, but also touted its 24-hour customer service network. It required an extensive capital investment to create, train, and maintain such a capable network and build out the logistics. Such folks commanded hefty fees. Xerox enjoyed a large revenue stream from the service of its copiers, which required highly trained and skilled technicians. The company was at the top of its game, a seemingly impenetrable fortress with a sizable moat around it. In the same way that "to google" has become synonymous with "to search the web," people didn't copy documents, they Xeroxed them.

But within five years, the company's market share fell from 95% to 14%. By the end of the decade, profits from Xerox's \$7B copying business had sunk by 40%. Today Xerox represents 17% of the market it once dominated. What happened?

Canon entered the market.

Canon had dedicated technology research in the 1960s to develop an alternative to Xerox's patented photostatic copying process. To create what it called the "New Process," Canon drew on two of its existing capabilities and techniques: micro-electronics, which it knew from its existing calculator business, and optics and imaging, which it drew from its camera business. This allowed it to make smaller copiers.

Canon designed its copiers for high reliability. They had only eight basic parts, making them orders of magnitude simpler than Xerox's products. In a shocking move, Canon made the primary assembly (toner, copier drum, charger, and cleaner) to be disposable. This was unthinkable, that you would design a key component of a critical piece of technology to be disposable. But Canon had its customers in mind: customers could easily remove and replace the assembly. This meant there was no need to build out, train, and manage the logistics for an extensive service department, keeping Canon lean and its costs down.

Furthermore, Canon designed its copiers around the manufacturing process—an inversion of conventional wisdom. The copiers could be made by robots on an assembly line, which dramatically reduced production costs. This meant Canon could redefine the market: instead of having to make a product that only the richest and largest companies would need or could afford, Canon designed its copiers this way to capture the individual and small business markets, selling them for \$700 to \$1,200. This opened up a new revenue stream, one that the market leader could not compete with. It quickly eroded Xerox’s large corporate business, because companies realized they could have a hundred Canon copiers for the same cost, reducing their risk if anything went wrong, and they could budget for them much more easily.

This story illustrates how a technology strategy can work hand in hand with the business strategy, how they can drive as copilots. It represents a combination of technology and business strategy wonderfully aligned and interlinked. This is the essence of what a technology strategist does. With that in mind, let’s look now at the role of the architect followed by the role of the strategist.

## The Architect’s Role

There are two jobs in the world that people want to do the most while knowing the least about: architect and strategist.

I should start by saying that this section does not offer a treatise on how to do architecture. I’m offering an overview of my perspective on the field, which I hope is a unique and interesting take on it, in order to provide context for the work at hand: devising a winning technology strategy for your business.

Technology systems are difficult to wrangle. Our systems grow in accidental complexity and complication over time. Sometimes we can succumb to thinking that other people really hold the cards, that they have the puppet strings we don’t.

This is exacerbated by the fact that our field is young and growing and changing, and we’re still finding the roles we need to have to be successful. To do so, we borrow metaphors from roles in other industries. The term “data scientist” was first used in the late 1990s. In 2008 or so, when “data scientist” emerged as a job title, it was widely ridiculed as a nonjob: the thought that people who just

worked with data could be scientists, or employ the rigors of their time-honored methods, was literally laughable in many circles. By 2012, *Harvard Business Review* published an article by Jeff Hammerbacher (of Facebook and Cassandra fame) and DJ Patil called “Data Scientist: The Sexiest Job of the 21st Century.” Today, it’s one of the most desired jobs, with pundits declaiming the terrifying state that we do not have nearly enough of them to tackle our most central technology problems.

Likewise, the term “architect” didn’t enter popular usage to describe a role in the software field until the late 1990s. It, too, was ridiculed as an overblown, fancy-pants misappropriation from a “real” field. Part of the vulnerability here is that it hasn’t always been clear what the architect’s deliverables are. We often say “blueprints,” but that’s another metaphor borrowed from the original field, and of course we don’t make actual blueprints.

With such origins, and with the subsequent division of the architect role into enterprise architect, solution architect, data architect, and so forth, the lines have blurred further. The result is that decades later, the practice and the art of the architect in technology varies dramatically not only from one company to the next, but also from one department and one practitioner to the next.

So we will define the role of the architect in order to proceed from common ground. This is my tailored view of it; others will have different definitions. Before we do that, though, let’s cover some historical context that informs how we think of the role.

## Vitruvius and the Principles of Architecture

Architecture begins when someone has a nontrivial problem to be solved. The product management team states *what* must be done to solve the problem, and the architect describes *how* to realize that vision in a system.

The first architect of record is a fellow named Vitruvius, who worked as a civil engineer in Rome in the first century BC. While you may not know his name, during the Renaissance, Leonardo da Vinci popularized the “Vitruvian Man” with perfect proportions based on Vitruvius’s ideas. Everyone who goes to architecture school learns his work.

Vitruvius is the author of *de Architectura*, known today as *Ten Books on Architecture*. It's a delightful, engaging read, and had a strong influence on Renaissance artists such as Michaelangelo as well as da Vinci. In it, Vitruvius expands on the three requirements that any architecture must demonstrate:

*Firmitas*

It must be solid, firm.

*Utilitas*

It must be useful, have utility.

*Venustas*

It must be beautiful, like Venus, inspiring love. This is sometimes translated as “delightful.”

It's a given that we must design a system, including a local software architecture, that actually runs, that it's “solid.” It may need to run for many years, even decades, and be maintainable to adapt to changes over that time. Solid doesn't mean inflexible. Skyscrapers are built on purpose to sway slightly with the wind, specifically to be more durable. The Sears Tower in Chicago regularly sways between six inches and a foot; taller buildings in America sway as much as four to five feet. Your architectures, and your strategies, must be similarly flexible in order to endure. We'll look at this later when we discuss how to support evolutionary architectures through our strategies.

It must also be fit to purpose, which means understanding deeply what the real purpose of the system is, and how to manage user expectations. This is supported in real terms through standards and consistent application of conventions, both in the information architecture (i.e., the user experience and design), and within the software construction itself.

Beauty, for Vitruvius, isn't really in the eye of the beholder. It is about harmony of proportion. One suggestion we can deduce from this for our current purposes is that we must rightsize our architecture and strategy work for the task at hand.

Vitruvius states—without irony—that an architect must concern himself with and become educated in several diverse fields of study, such that they find their way into the work. He outlines them in Chapter 1 of *de Architectura*:

- Skill in manual labor as well as in theory
- Proclivity and desire for continuous learning
- A dexterity with tools
- An understanding of optics—how the light gets in
- History, such that you can emphasize and not misinterpret signs of cultural significance
- A strong understanding of philosophy, in order to practice abstract thinking as well as honesty and courtesy
- Physics, to help make things sturdy
- Art, music, theater, drawing, painting, and poetry, to help make things beautiful and well suited to their human purposes
- Math
- Medicine
- Astronomy
- Politics

He concludes that absent a degree of education and even lay practice in any one of these areas, one cannot refer to oneself as an architect. These are excellent guides for us in technology today. For those of us concerned with the business of making software and setting the direction for other technologists, to hold ourselves to account in these ways would serve us very well.

In a recent conversation I had with Ben Pring, philosopher, noted futurist, and director of The Future of Work Center at Cognizant, he underscored the importance of beauty in software, pointing out that historically our most culturally significant buildings have been not merely adorned, but specifically built with beauty in mind as a central, driving narrative. I conclude from this that such foregrounding reinforces in the popular imagination the power of the institutions that build them. I base this conclusion on the preface in the *Ten Books*, in which Vitruvius writes openly and directly to Emperor Caesar, stating:

But when I saw that you were giving your attention not only to the welfare of society in general and to the establishment of public order, but also to the providing of public buildings intended for utilitarian purposes, so that not only should the State have been enriched with provinces by your means, *but that the greatness of its*

*power might likewise be attended with distinguished authority* in its public buildings, I thought that I ought to take the first opportunity to lay before you my writings on this theme. (emphasis mine)

Realizing these broad dicta into an architecture means, I think, finding the concentrations of power, and determining how to best support and ultimately inspire the human factor in the forms we create. I hope once you're done with this book, you'll have some ideas for how to enable and reveal the three facets of *firmitas*, *utilitas*, and *venustas* in your own work.

## Three Concerns of the Architect

Whereas developers are typically focused on delivering working code for a user story within the next two weeks for one system within their one team, architects are concerned with how technology can fulfill business goals given a long-term outlook across a variety of interrelated systems across many teams. It's analogous to a project view versus a portfolio view. They should have their visors raised much higher. The architect is hopefully not concerned with low-level details of the code itself inside one system, but is more focused on where data-center boundaries are crossed, where system component boundaries are crossed.

Here's my definition of an architect's work: it comprises the set of strategic and technical models that create a context for position (capabilities), velocity (directedness, ability to adjust), and potential (relations) to harmonize strategic business and technology goals. Notice that in this definition, the role of the architect and technology strategist is not to merely *serve* the business but to play together. I have been in shops where technology was squarely second fiddle, a subservient order-taking organization to support what was deemed the real business. That's no fun for creative people who have something to contribute. But more importantly, I submit that businesses, now more than ever, cannot sustain such a division, and to create greater competitive advantage must work toward integration with co-leadership.

Over my 20 years in this field, I've come to conclude that there are three primary concerns of the architect:

- Contain entropy.
- Specify the nonfunctional requirements.

- Determine trade-offs.

There are many different roles that architects legitimately play in different organizations. But the primary struggle I have seen comes when they are not focused on a deliverable, on what could be conceived as a “blueprint.” Without that focus, they tend to weigh in at project meetings or make declarations informally that can’t be remembered or followed. To stay pertinent to the project, and to help guide it in a way that others may not have the purview to do, drawing a line at these boundaries seems to work out pretty well. The definition remains, of course, rather open to interpretation, in grudging deference to the machinations of the real world.

Let’s unpack each of those responsibilities.

### **Contain entropy**

This viewpoint on the architect’s work I learned in a fun conversation over dinner in New York with the very smart and funny Cameron Purdy, the founder of Coherence, who at the time ran Java at Oracle. “Entropy” refers to the second law of thermodynamics, which roughly states that systems over time will degrade into an increasingly chaotic state, such that the amount of energy in the system available for work is diminished.

The architect defines standards, conventions, and toolsets for teams to use. These are common practices, and generally idiosyncratic to any given organization. As application or solution architects, they help within a system, within an ecosystem, and across an organization to create a common set of practices for developers that help things both go quicker and be more understandable and maintainable. This is a form of containing entropy. As we mature, we realize that picking one tool or framework or language or platform is not a matter of personal taste, but rather a choice with broad ramifications for future flexibility, mergers and acquisitions, training, our ability to hire future supporting teams, and our future ability to directly support—or subvert—the business strategy.

Those with more business-oriented concerns and technologists cannot ignore each other’s fields. Working as a pattern-maker and a synthesizer, the architect-as-strategist broadens and ennobles these concerns, creating technology strategies that both are rooted in the causes and concerns of the business and recognize its constraints and opportunities. In collaboration with product management, and

with colleagues in strategy, business development, finance, and HR, the architect works to ensure that there is alignment between the systems, yes, but also between those systems and the organization, and between the organization and its stated aims.

In short, for far too long we architects have thought we were in the business of making software. But we're in the business of building a business.

The architect who is containing entropy is stating a vision around which to rally; showing a path in a roadmap; garnering support for that vision through communication of guidelines and standards; and creating clarity to ensure efficiency of execution and that you're doing the right things and doing things right.

I love this definition of containing entropy because it offers something to both the software-minded and the business-minded architect (which I hope are two categories this book will help collapse). One cannot be successful as an architect without thinking of not only *what* to do, but *how* to get it done within an organization, which requires knowing *why* it should matter to someone who isn't a technologist.

We often hear of architects with failed dreams of how the system should have been. They are consumed by writing documents and those documents are subsequently ignored, leading them to give up. Left with only the most informal conversational avenues to offer insufficient direction to teams, they become frustrated and even marginalized.

Knowing that you're in the business of building a business, and that technology is just an avenue by which you enable that, is a critical first step to being not only useful but powerful as an architect and strategist.

### **Specify nonfunctional requirements**

Knowing what you're on the hook for, letting others know it, and making sure that it's a concrete deliverable will all go a long way to ensuring your vision is understood and realized.

Product management is responsible for specifying what the system must do for the end user. They might state functional requirements in user stories and epics.



The nonfunctional requirements are properties of the system that do not necessarily appear directly to the user. They are typically described as the “-ilities.” The ones I focus on most are scalability, availability, maintainability, manageability, monitorability, extensibility, interoperability, portability, security, and performance.

The architect is responsible for specifying how the system will realize the functional and nonfunctional requirements in its construction. In order to do so, she must write a document that specifies how these will be realized.

This document, the *architecture definition*, serves as the technologist’s answer to the blueprint. It should be structured in four broad categories to include business, application, data, and infrastructure perspectives, and expressed with clarity and decisiveness, using primarily testable statements as valid propositions (which we’ll examine in the next chapter) and math.

Finding ways to make those expressions concrete and executable is too often overlooked. In addition to writing and publishing a formal architecture definition document to the teams, you can do this by adding nonfunctional requirements to user stories as acceptance criteria.

### Determine trade-offs

*You can never try to escape one danger without encountering another. Prudence consists in recognizing the different dangers and in accepting the least bad as good.*

—Machiavelli, *The Art of War*

As we know, every action produces an equal and opposite reaction. Adding security reduces performance. Sharding and partitioning the database affords greater performance and distribution but creates complexity that is difficult to manage. Adding robust monitoring can generate huge volumes of log data to be stored, rotated, secured, and cleansed. Keeping the design “simple” often defers the interests of flexibility until later, where it becomes very expensive.

The role of the architect is to see where those challenges may lurk, seek to make them explicit, and make value judgments about how to balance the solutions and the new problems they occasion, under the guidance of the broader business strategy. As English poet John Milton wrote in *Paradise Lost*, you make “the darkness visible.”

In short, you're never quite solving a problem. You're only trading it for one that you'd rather have. We solve our need for shelter by assuming a mortgage that we then must pay for. Paul Virilio, the French cultural theorist and philosopher, reminds us lucidly, "When you invent the ship, you also invent the shipwreck...Every technology carries its own negativity, which is invented at the same time as technical progress" (*Politics of the Very Worst*, Semiotexte). Your architecture and strategy work will do well to examine not only how you are addressing the problems you've been given, but also what new problems your solutions precipitate.

Any trade-off eventually reduces to a trade-off of time and money.

Absent a strategic mindset, many technologists left to their own devices create what amounts to little more than shopping lists of shiny objects. These can include the latest and most fashionable tech because it's popular or because it might bolster their résumé. We hear this frequently described as "a solution looking for a problem." Moreover, the less shallow or cynically minded among us are still rather prone to chasing exciting technology for its own sake, not unlike a dog chasing a squirrel. Intellectual curiosity is a wonderful thing, a best thing. But to ensure that your technology and architecture decisions are truly supportive of the business—that is, give it the best chance to create competitive advantage—they need to be not shopping lists of shiny objects, but squarely *strategic*.

So let's look at the role of the strategist.

## The Strategist's Role

*Strategy is about getting more power than the starting position would suggest. Strategy is the art of creating power.*

—Lawrence Freedman, *Strategy: A History* (Oxford University Press)

The word *strategy* originates from the Greek *strategos*. The term first appeared in fifth-century Athens as a conflation of the words meaning the expansion of the military general, and came to be used to refer to the offices or science of the general—the general's work. But the word *strategy* entered general use only at the start of the 19th century in Antoine-Henri Jomini's writing on Napoleon's methods.

Jomini was of Swiss origin; he started out as a banker in Paris, later joined the French army under Napoleon, and eventually got promo-

ted to general. Jomini began writing down Napoleon's methods in such a lucid manner that they came to be published as a book, entitled *Treatise on Major Military Operations*, in 1803. Jomini's strategies were employed in the US Civil War and eventually taught at West Point Academy. He is considered the founder of modern strategy by many military historians.

Jomini's definition of strategy helpfully divides the word. He writes, "Strategy decides where to act; logistics brings the troops to this point; tactics decides the manner of execution." In other words, means (resources) are allocated and subjected to a method in order to achieve a goal.

Yet definitions of strategy vary. One of the more abstract definitions comes from Sun Tzu, a Chinese general and philosopher, and author of *The Art of War* in 500 BC. His book was not translated to English until the 20th century, at which point it began serving as a foundational text for guiding military strategies. It entered the popular imagination once it got adapted and marketed for business purposes.

He writes, "Strategy is the art of making use of time and space." This is a tall order, and while aesthetically I appreciate the definition, we can break this down further in order to come to something practically executable.



### The History of Strategy

If you're interested in the intellectual history of strategy, its origins, and its evolution from military thought to game theory to business, I highly recommend Lawrence Freedman's *Strategy: A History* (Oxford University Press). It's a fascinating read, and offers a much richer view than we need here.

For our purposes, strategy is about determining the problems and opportunities in front of you, defining them properly, and shaping a course of action that will give your business the greatest advantage. Balancing problem solving with creating and exploiting new opportunities through imagination and analysis is the cornerstone of a great strategy.

Echoing Jomini, we'll say that strategy is about determining the best balance between a set of goals, the method used to achieve them, and the resources available as means. With the current rate of change in business, we can't set it and forget it, expecting that a three- or five-year strategy will go unrevised. At the same time, constant revision amounts to a reactionary collection of tactics, which is no strategy at all.

Most business strategies will concern themselves with the following:

- The goals of the organization
- The operating model: processes and how your company conducts its business
- Culture: the mores and value system, the modes of communication
- Talent strategy: how you source and retain talent, how you train them
- Facilities strategy: where you do business, relevant local laws, and cost concerns

Strategies should be created at different levels: broad corporate-level strategies, business unit or division strategies, departmental strategies, and portfolio strategies. These will be more or less formal, and be revised more frequently according to the climate and what you find yourself in (see “[Life Cycle Stage](#)” on page 111). (Life cycles are discussed in [Chapter 5](#).)

## The Triumvirate: Strategy, Culture, and Execution

*Culture eats strategy for breakfast.*

—Management professor Peter Drucker

Any business aims to do one or many of these things:

- Grow shareholder value
- Grow earnings per share
- Increase revenue
- Manage costs
- Diversify or create new revenue streams
- Cross-sell more products

- Increase market share
- Increase share of wallet
- Increase yield
- Improve customer retention
- Reduce product error/defect rates
- Improve safety
- Improve time to market/speed of operations
- Grow through acquisition

Of course, there are different emphases at different times. To achieve these aims, broadly speaking, the strategist asks these questions:

- Are resources devoted to the right areas, to the most important customers?
- Are we creating products and services that can thrive in a market in different time horizons?
- Where should we spend money? Where should we cut costs?
- Where do skills need to be added or strengthened?
- Where can productivity be improved?
- What culture, attitude, and skills are required?

Many companies have a Chief Strategy Officer or VP of Corporate Strategy. Strategy season frequently begins in the spring, giving this person and her team a couple of months to prepare a deck to present to the executive leadership team in the late summer. This will be discussed, revised, and eventually approved and used as input for budget season, which begins in the fall and continues until the budget for the following year is approved. We in technology tend to like to see our ideas realized moments after we have them. Being aware of this calendar and corporate planning process will help you plan for adding any big-ticket items to the slate in time for them to receive the necessary attention, support, and budget allocations.

That said, the evolution of agile software methods, the preponderance of “disruptive” startups, and a growing global economy have all aligned variously to dilute the formality and rigidity of the strategist’s role in such a process, leading her to rely more on regular con-

versations with the executive team, and create reports with tighter scopes on an ongoing basis.

Depending on her level of power and position within the organization, the strategist finds herself concerned with some or all of the following:

- Identifying business development opportunities, such as partnerships, joint ventures, cooperative arrangements with competitors, and the like
- Finding, proposing, and validating mergers and acquisition opportunities
- Building strategic capabilities within certain areas of the organization, such as helping create a sustainable AI practice in the face of growing trends
- Performing research based on data to recommend long-term directions for the company (generally 1–3 years)

This last one is very common, and how many strategists are trained as consultants entering the field at the venerable strategy firms such as Bain, Boston Consulting Group (BCG), and McKinsey. They likely work with business analysts, marketing, sales, technology, and operations teams in a cross-functional working group to develop hypotheses for how the business climate might be enabling or impinging upon their competitive advantage, and how they should define a goal and direction and allocate resources to win in the marketplace.

According to one [McKinsey report](#), 40% of strategists responding to their survey are most focused on “using fact-based analysis to spot industry shifts and to understand their own companies’ sources of competitive advantage as a foundation for clear, differentiated strategies.”

But spending months researching and creating data-driven decks is no longer enough. Because the world is moving so fast, the traditional strategist has taken the driver’s seat in building capabilities. As the walls between business and technology continue to fold in on each other, the strategist may well find himself leading a team of data scientists to create an analytics platform to help themselves and customers gain precious insights into their business operations. My colleague Balaji Krishnamurthy, the VP of Strategy at Sabre Hospi-

tality, who was previously in strategy roles at McKinsey and LinkedIn, offers this observation:

To be a good strategist, you need to be ready to deal with ambiguity. You need to be ready to pivot. You must form a hypothesis quickly about what must be done, then synthesize lots of data. You must then see options and possibilities available, determine a goal, and present your findings clearly with a recommendation on how to allocate resources to achieve that goal.

Ultimately, companies are looking to grow and gain some distinctive competitive advantage. They can do this through technical innovation properly applied to real-world business problems. One assertion of this book is that the roles of Chief Architect and Chief Strategist are more blurred, and more aligned, than ever, and that their mutual understanding of each other's concerns and methods will be an increasingly important driver for winning organizations.

Learn from your executive and product leadership teams what areas of focus they have for their business strategy and product roadmaps, so you can be prepared to match your technology to them.

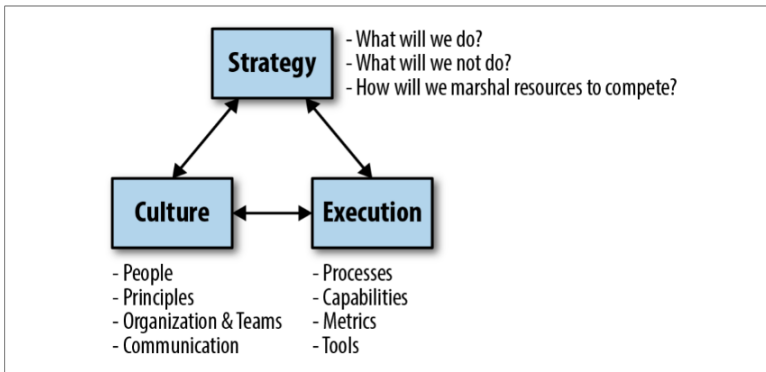
For example, if your business is in cost-cutting mode, as companies tend to be when revenue is soft or they're preparing for an IPO, then your technology strategy should match. You can do that by examining the people angle: Can you move workers or ramp down in expensive cities in order to hire programmers in lower-cost development centers? Can you examine your delivery and release processes to add automation and reduce manual labor there? Can you use free and open source libraries in place of expensive commercial software? These are examples from people, process, and technical perspectives of how you can map your technology strategy to the business strategy.

A *Strategy Deck* is analogous to an architecture definition document for the organization. Neither will achieve the desired aims if you assume it lives in a vacuum.

The culture of your organization comprises your stated principles, and to a far greater extent, the actual lived principles as reflected by the attitudes, communication styles, and behaviors of your teams. If your teams are territorial and competitive, an integrative platform strategy must identify and address that challenge.

Finally, your teams must be ready and capable of executing on your strategy. A Strategy Deck that states lofty, exciting aims will fail if it

also doesn't include diligent, consistent execution and clear metrics to measure its success. This triumvirate is illustrated in **Figure 1-1**.



*Figure 1-1. The triumvirate dominating forces of strategy, culture, and execution*

Find ways to work with your leadership and across teams to ensure all of these forces are aligned. A good first step for doing so is to create two versions of the strategy: one that provides an honest and detailed examination of all three factors to share with the executive team, and another shorter version that communicates only the changes you're driving in a way that you can share publicly with teams. In long-range planning there are financial, business transaction, and personnel matters that obviously can't be disclosed.

## Summary

In this chapter, we defined the roles of the architect and strategist. We highlighted key responsibilities and practices for these fields and set the context for the next part, in which we begin our exploration of the pattern language, starting with strategy creation patterns.





---

# Creating the Strategy

Part II of this book is organized around the “creation” patterns—those that help you define components of your technology strategy.

## A Logical Architecture of the Creation Patterns

Here I present 19 creation patterns to help you turn a vision into a holistic, strong technology strategy. They bridge the gap between an idea and an executable plan that has taken all key aspects into account.

The following is a kind of logical architecture of the 19 creation patterns. They are divided into five categories, starting with the broadest scope and narrowing from there, each of which corresponds to a chapter in this part:

- Analysis
- World
- Industry
- Company
- Department

**Figure II-1** illustrates the logical architecture of how these patterns work together.

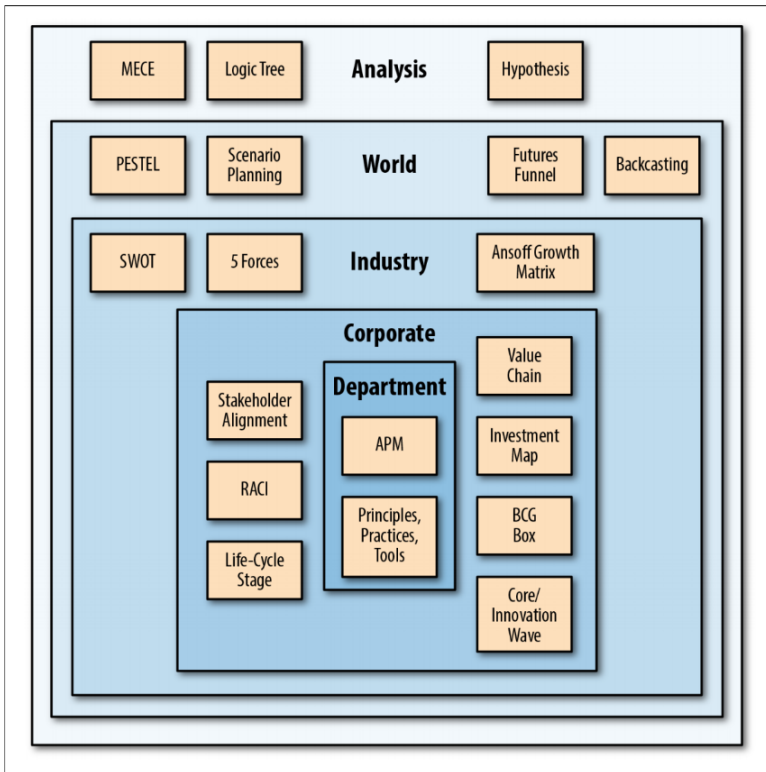


Figure II-1. The set of creation patterns all together in a logical architecture

This architecture is organized into concentric shapes representing increasingly narrow scopes of influence, impact, and relevance. This should help you to match the problem to the set of applicable patterns. The divisions are not strict; they're just a guide. Consider this a visual table of contents for all the patterns relating to creating your technology strategy. It's meant to act like a high-level, logical system architecture. Its purpose is to offer you an overview of the patterns all together and show where they operate in their different spheres. It gives you a quick reference point to select the tool that might help you for the job at hand.

Your boss might simply ask you to “make a technology strategy,” because she felt she didn't have one, or didn't know what it was, and thought it would be a good idea to create one. This has happened to me several times in my career. In these cases, you need to ask clarify-

ing questions about the concerns she expects you to address, and the scope of the recommendations she expects, so that you aim at the right level.

Consider the following situations.

You need to make a strategy to upgrade the online database. In this case, you've got a clear, localized, specialized problem. But it may have high impact if things go wrong. So you'll need a plan, and you'll need to contact some folks in different organizations if you miss the cutover window. So maybe this is a small project, and you only need to employ the MECE, Logic Tree, Stakeholder Matrix, and RACI patterns, along with elements of Principles, Practices, and Tools, to make your plan.

Or, you've been asked for a strategy to migrate from your home-grown legacy billing system to a new off-the-shelf product. This is a larger project that has impact and constraints outside your tech department and requires more analysis and understanding of your application portfolio and trends in the industry.

Maybe your boss asked if your software development method is right, or if you should switch to a new architectural model. These are larger problems, and you should expect to employ many or all of the patterns in the scopes of Department, Company, and Industry.

If you're the CTO, CIO, or Chief Architect, or you're on the enterprise architecture team and have been tasked with creating the technology strategy for your company, expect to employ all the patterns here. This work will take weeks and involve a lot of reading and writing and refining.

You don't design software by opening the classic *Design Patterns* book and dutifully making one component for every pattern. It's the same here. Use what pattern you need when you need it, and they'll be different for different strategy problem scopes.

But unless you're creating an overarching, multiyear strategy for your entire technology organization, you don't need to use all of them all of the time. That would be overkill, incredibly time-consuming, and silly. Of course, keep this book around as a reference and use the ones relevant for the scope and type of the job at hand.

Now let's look at the patterns.



## CHAPTER 2

---

# Analysis

*To use language is to enter into the territory of categories, which are as necessary as they are dangerous.*

—Rebecca Solnit, *The Mother of All Questions*

*The only cost that matters is opportunity cost.*

—Larry Page

This chapter covers foundational patterns for analysis that you can broadly apply. They are *MECE*, *Logic Tree*, and *Hypothesis*.

*MECE* stands for “Mutually Exclusive, Collectively Exhaustive.” It represents a kind of metapattern. It offers a quick way to check that the building blocks of your strategy work are valid and complete. I call it a metapattern because it doesn’t produce any direct output that you can drop right into your strategy like many of the others. It’s a light form of analysis that’s broadly applicable across all the other patterns we’ll explore.

*Logic Tree* is used by strategy consultants as a simple tool for determining a set of relevant problems and possible causes. It helps organize your ideas, making quick work of examining any problem.

The *Hypothesis* pattern is a way of making a guess, based on some supporting suppositions and data, about what the root problem might be.

The patterns we’re starting with are the most abstract. These are tools for analysis that will act as the underpinnings of any strategy work.

In the world of strategy consulting, analysis of this kind is performed on what they call cases. A *case* is a particular industry problem to be solved, like a detective “on the case.” Job candidates for consultant positions at McKinsey or Bain or BCG must go through the case interview, in which they use a framework of tools and a certain approach to analyzing a problem to properly define and understand it, so they can make good recommendations or solutions. This is not dissimilar from when we are asked to envision a project or an architecture or define a technology solution within a business context. It’s about how to make great choices from competing viable alternatives.

In business, *opportunity cost* refers to what happens when you pick one alternative from many: you may realize a gain from the one you pick, but forfeit any potential gains that could have been realized by the opportunities you didn’t pick. If the returns on the choice you picked are more than the returns you could have had otherwise, you made the best decision from the available options.

There are obvious questions we get asked a lot. Do you choose to upgrade the current data center, or move to the cloud? Should you build or buy? Should you train your teams on artificial intelligence in-house, or execute an “acqui-hire” (buy a company not for its technology or customers but to get its knowledgeable employees)? What database vendor should you go with? Do you rush to be first to market and get customer feedback even if your product is a bit buggy, or make it solid and delay the launch?

Answering these questions well is hard, because these are complex problems with many moving parts, and because there is considerable risk involved when decisions are hard to reverse, when they’re costly, or when you get only one shot at them. As architects, we’re asked to make recommendations with imperfect knowledge, and need to do research, try some stuff out, and make a call. The more times we show good judgment, make the right call through a fog of business uncertainties, and minimize opportunity cost and maximize returns, the more our own stock price goes up in the organization.

As a technology strategist, you have many jobs:

- Survey the landscape across your industry, organization, customers, stakeholders, competitors, and employees.

- Examine trends in technology.
- Determine what current priorities, problems, and possible opportunities are presented to your company.
- Analyze and synthesize these problems and opportunities into a course of action: decide what to do, and what not to do.
- Make strong recommendations for how to allocate your company's resources, in what way, in what places, to what extent, and to what end.

That is the work of the technology strategist, whether you're an architect, director, VP, CTO, or CIO.

Because there is not unlimited money and time to invest in everything, strategy is about making the right recommendations to minimize organizational damage and positional disadvantage, and maximize advantage, profit, and benefit. The better you are at raw analysis, the more often you'll make choices with higher probabilities of winning.

## MECE

MECE, pronounced “mee-see,” is a tool created by the leading business strategy firm McKinsey. As stated previously, it stands for “Mutually Exclusive, Collectively Exhaustive,” and dictates the relation of the content, but not the format, of your lists. Because of the vital importance of lists, this is one of the most useful tools you can have in your tool box.

The single most important thing you can do to improve your chances of making a winning technology is to become quite good at making lists.

Lists are the raw material of strategy and technology architecture. They are the building blocks, the lifeblood. They are the foundation of your strategy work. And they are everywhere. Therefore, if they are weak, your strategy will crumble. You can be a strong technologist, have a good idea, and care about it passionately. But if you aren't practically perfect at list-making, your strategy will flounder and your efforts will fail.

That's because everything you do as you create your technology strategy starts its life as a list, and then blossoms into something else. Your strategy is, at heart, a list of lists. Thinking of your work



from this perspective is maybe the best trick to creating a sane, organized, productive context for your work. Let's talk about lists for a moment.

There are two parts to a practically perfect list: it must be conceived properly, and it must be *MECE*, which we will define in a moment.

In a properly conceived list, two things are crystal clear:

- Who the audience is
- Why they care

You can determine who your audience is by asking the following key questions:

- Upon reading this list, can the audience make a decision they could not make before having the information in the list?
- Upon reading the list, can the audience now go do something they could not have known to do before?

These are the two reasons to bother creating any kind of information in a strategy. In this context, there is little point, time, or patience for a document that merely helps a general audience “understand” something. Your lists must be lean. That means making them directive toward work that someone will go and do, or providing the data that allows a decision maker to decide the best course of action. The RACI is a list. It answers the question for the project team of who is assigned to what role so that everyone knows who is in charge of what, who is the decision maker for what, and who is doing the work, and if someone sees his name on the list with an “R” by an item, he can go do that work. The Stakeholder List is primarily for the project manager. It lets him decide whom to include in what meetings and whom to contact for certain questions. But if these, and all the many other lists you create as part of your technology strategy, are not *MECE*, your building blocks will be weak and your strategic efforts will crumble. Let's look at some examples to make this clear.

This formula is *MECE*:

*Opportunity Cost = Return of Most Lucrative Option – Return of Chosen Option*

This formula is MECE:

$$\text{Profit} = \text{Revenue} - \text{Cost}$$

Revenue – Cost = Profit is MECE. That’s because together those three items make a complete thought, divided across lines that don’t overlap, and nothing is left out. All of the parts of the money are accounted for within the same level of discourse. It is nonsense to leave out Revenue and simply state “– Cost = Profit.” There are only two ways to increase profit: increase revenue or decrease costs. Recognizing the formula as MECE can help remind you to address both the cost and the revenue aspects in your strategy.

This list is MECE:

*Spades, Diamonds, Hearts, Clubs*

This list is MECE:

*Winter, Spring, Summer, Fall*

Each entry in the list is *mutually exclusive* of every other one. There is no overlap in their content. Winter ends on a specific day of the year, and then the next day is the start of Spring. Every date on the calendar is, with certainty, part of one and only one season. There is no card in the deck that is part Spades and part Diamonds.

The elements in the list, when taken together as a collection, entirely define the category. No item is left out, leaving an incomplete definition. Thus, the list is *collectively exhaustive*.

This is not MECE:

*North, South, West*

It’s not collectively exhaustive. It fails to include East, and is therefore an improperly structured list.

Consider the following list:

*Revenue – Cost = Profit. Free Cash Flow.*

This is not MECE because “free cash flow” is not at the same level of discourse as the other items. It is true that free cash flow is an important part of any public company’s earnings statements. But that is unrelated to this equation, even though they appear to all be in the category of “stuff about money in a company.” That’s a weak

category for a list because it's not sufficiently directed to an audience for a goal.

What about this one:

*Internal Stakeholders, External Stakeholders, Development Teams*

This isn't MECE because "internal" and "external" divide the world between them. Development Teams are a subcategory of Internal Stakeholders for a technology strategy.

Elements that are subcategories of other elements must not be included. Consider this list:

*North, South, Southwest, East*

This is not MECE because it leaves out one of the elements, West, and so is not collectively exhaustive. It also includes Southwest, which is not topologically on the same plane as the other elements. It dips into a lower level of distinction, as in the "free cash flow" example. Southwest is contained within the higher level of abstraction of South. So the elements on this list are not mutually exclusive.

These examples are straightforward (obvious) in order to illustrate the point. But they share an attribute that precious few lists in the world have: they are enums by definition. It is clear what goes on the list and what doesn't. Most things in life are not this simple.

Consider the following list of departments or job roles in a dev shop:

- Software Developers
- Architects
- Analysts

It's not exhaustive: we left out Testers, and other roles depending on your organization, such as Release Engineers, Database Administrators, Project Managers, and so on. To test if our list is MECE, we must ensure we have pushed ourselves to think of all the relevant components that make up that category.

Remember the first rule: know your audience. Your longer, more detailed lists should be kept for your private analysis to help you reach your conclusion, or reserved for lists of things to be done in the project, such as a work breakdown structure. But you don't want long lists when working with executives because they have Executive ADD. Even though you'll worry that you're leaving crucial things

out, just give them the summary, but make it MECE. Then you can reveal only the headline: the impactful conclusion that makes a difference to your audience.

## The Rule of Three

A good rule of thumb is to find the level of abstraction that keeps your lists in categories of three or five items. For whatever reason, people seem to more naturally understand and remember lists of three, or at the least, odd-numbered lists. Consider two movie titles: *The Good, The Bad, and The Ugly* is more memorable than *The Cook, The Thief, His Wife, and Her Lover*. Push yourself to make your lists with three to five items. Prefer lists of three or five over lists of four. You'll find this little trick helps keep your thinking quick and nimble, and it will shorten your turnaround time because your work will be closer to what you'll need to present to executives and stakeholders.

Consider this list of age groups:

- 0–5
- 6–10
- 11–15
- 16–25
- 26–35
- 36–45
- 46–54
- 55–65
- 66–75
- 76 and above

This list is technically MECE. None of the categories overlap, and the sum of the subcategories equals the whole category. It might be OK for a data scientist doing customer segmentation. But probably not even then. It's too fine-grained and low-level, so it's not very good for strategy work. You need to keep your visor higher; look more broadly to horizons to distill the few things that really make an impact and drive change. It's more analysis and art than science. So even though the list is technically correct, you will lose your audience with details like this, and you can find ways to cluster and consolidate them better, along the contours of a real difference or

divergence depending on your own organization's products, services, and markets.

Let's look at a quick example of how to apply this idea of MECE lists.

## Applying MECE Lists

Imagine you've been enlisted to create a recommendation to the CTO for a new database system to replace your legacy system. If you merely state the single database system you want to buy, any responsible executive will reject your recommendation as heavily biased, poorly considered, and potentially reckless.

So we want to first consider our audience, with empathy, and always ask: Who is this for, and what do they need to know either to make a decision or to do the thing in question?

Your deciding audience wants to know that they have been given a clear, thorough, thoughtful, unbiased proposal and that they are not being manipulated. In our empathy, we realize that everyone has a boss, and that no one in a company of any size just makes a decision in a vacuum. It's not the CTO's money. So your CTO must in turn answer to his bosses for the system he selects, and is accountable for its success. Your recommendation will be successful if you give your deciding audience a list of MECE lists.

But the list of database system choices is potentially in the thousands. It is impossible to include all of them, and impractical and unhelpful to include even 20 of them. Being ridiculous is not what is meant by "collectively exhaustive." So first we'll create a list of criteria to help us make our final list MECE. I include three or five factors on which you will base your selection and write those down, as they become part of your recommendation too. You're showing your audience how you came to your conclusion, just like showing the long math in school: you're not just giving the answer, but providing the steps by which you arrived at it. This helps the audience follow your story and agree with your conclusion.

Then we'll perform a survey of the landscape, including systems that meet the criteria. Include open source alternatives as well as commercial vendors. We might have a few of each. If we recommended only the one we already wanted, we would miss the chance to perform the analysis, squander an opportunity for learning that might change or augment our view, and lose confidence in our choice and

ability to execute. Including only our one recommendation would certainly and immediately invite considerable skepticism and questioning about the alternatives and how we considered them.

So make a MECE list of options. The list is exhaustive according to your chosen criteria. Say you have 8 or 10 options in your list of “all the database systems considered.” Say so in your recommendation. It shows you’ve done your homework and suggests less bias and a more data-driven, analytical approach. Then say you narrowed it down to five options to present. That list includes two you reject and state why. You have a list of three options remaining.

For each element on your list of remaining recommended vendors, create another list of lists: “advantages, disadvantages” (that’s a MECE list itself). The elements in each list should be something about the technology, particularly 1) the functional requirements such as key features that distinguish it from the competition and 2) nonfunctional requirements such as performance, availability, security, and maintainability (that’s a MECE list, too). Consider these systems also from the business perspective: ability to train the staff, popularity/access to future staff, ease of use, and so forth.

Then from the list of acceptable candidates, present them all, ranked as Good, Better, and Best. (The Good, Better, Best list is MECE too, because you wouldn’t improve its MECE-ness by adding a “Horrible” option: the category or name of this list is the *acceptable options*, which presumably does not include “horrible, and therefore unusable ones.”)

The Good option might be the one that is acceptable to you, and is low cost but not optimal. The Best one might be the most desirable but highest cost, and so on.

Organizing your list this way makes an executive feel more confident that you have an understanding of the entire landscape, aren’t too biased, and show your reasoning. That makes your recommendation stronger.

### **The Celesital Emporium...**

In 1668, English philosopher John Wilkins published a proposal for adopting a universal language as well as a universal system of measurements. In his estimation, this was an entirely rational classification system.

In 1952, Argentine poet Jorge Louis Borges published an essay titled “The Analytical Language of John Wilkins.” As a critique of Wilkins’s work, Borges offered the following list, in his story “The Celestial Emporium of Benevolent Knowledge,” purported to have been created by a 14th-century Chinese emperor as his taxonomy for classifying the members of the animal kingdom:

1. Those that belong to the emperor
2. Embalmed ones
3. Those that are trained
4. Suckling pigs
5. Mermaids (or sirens)
6. Fabulous ones
7. Stray dogs
8. Those that are included in this classification
9. Those that tremble as if they were mad
10. Innumerable ones
11. Those drawn with a very fine camel hair brush
12. Et cetera
13. Those that have just broken the flower vase
14. Those that, at a distance, resemble flies

The list is hilarious, because it is so obviously an example of an incomplete set of sets. There’s a lot left out here. Many of the categories also overlap (can a creature not be at once “fabulous” and belong to the emperor and have just broken the flower vase?). Do not all animals, at a sufficient distance, resemble flies? What belongs in “Et cetera”? Who could possibly make meaningful use of this?

Borges’s point was that there is not a single, unifying, rational way to classify All The Things, that there are cultural differences that affect our views, and that ultimately such taxonomies can be shown to be arbitrary. So that’s understood. The point here is that the division of animals in the “Celestial Emporium of Benevolent Knowledge” is perhaps the least-MECE list in the history of earth. Yet how many of our project and architecture lists, on further inspection, perhaps resemble it?

Getting good at quickly checking if you are thinking in lists and then making sure they’re MECE has the pleasant side effect of helping build your powers of analysis. Think of MECE as a lens. Every time you make a list, immediately test if it is MECE. Use it as a heu-

ristic device with your team: inspect your list with the team as you're meeting, be sure to ask if the current list you're working on is MECE, and then refine it. Your team may groan at first, but they will gradually start to see the value, and then they will not be able to imagine how they ever lived without it.

Make your work lists of lists, and make those lists MECE. Your recommendations have a better chance of getting accepted, supported, and executed on. And you will create more power for your organization and your team.

## Logic Tree

*If I had only one hour to solve a problem, I would spend up to two-thirds of that hour in attempting to define what the problem is.*

—Unnamed engineering professor at Yale, via William Markle

A *Logic Tree* is sometimes called an *Issue Tree* in the world of strategy consulting. The tree branches out as a decomposition of the problem you're starting with. Collect possible root causes into groups, using the MECE technique, and then break them down into subgroups. As a technologist, analysis of this kind should be very straightforward for you.

The output of a Logic Tree exercise is a diagram. You can draw it in a mind mapping tool or presentation software. If you sketch on a whiteboard or paper for your initial draft with your team, transfer it into digital form so you can keep it in your growing Strategy Deck.

You will use Logic Trees in two ways. The first is for determining the problem. These are called *Diagnostic Logic Trees*. The second is for determining the solution set, called *Solution Logic Trees*. Either way, you're following the same method with the same type of diagram as output.

Every strategy starts with a set of problems to be solved. The strategy itself is the set of solutions to those problems. A Logic Tree is the critical starting point for any strategy. It ensures you have defined the problem correctly and helps you enumerate the best strategic solutions.

If you are not very clear on the reason for making a strategy, it will be more general work, less relevant to any audience, and less executable. So if you're asked to make a multiyear strategy, or a smaller



local strategy, be certain you have alignment on what problem your strategy is meant to address before doing any other work. If you just got asked to make a strategy (as sometimes happens), be sure to ask your manager a few questions first about what problem she wants solved.

People at large organizations spend a lot of time doing hard work on poorly defined or unimportant problems. The result is useless at best, and a disaster at worst. To avoid this trap, you first must know what problem you are solving. There is no generic, cookie-cutter strategy in the world: there are frameworks to help you consider which set of actions is right for you. This one will help focus your work, make it go quicker, and make your resulting strategy more relevant and executable.

## Diagnostic Logic Tree

Diagnostic Logic Trees attempt to determine the applicable subcategories of problems and a root cause. They answer the question of *why* the issue has occurred.

As you ask “why,” you are using your powers of deductive reasoning, working backward from a known current state.

To reiterate, you start any strategy by first clarifying what problem you need to solve. You are then ready to create the Diagnostic Logic Tree to determine why this problem or situation is occurring.

## Solution Logic Tree

Solution Logic Trees are a way of representing possible solutions or courses of action to address a problem. They answer the question of *how* to proceed. You create this kind of tree after making the Diagnostic Logic Tree.

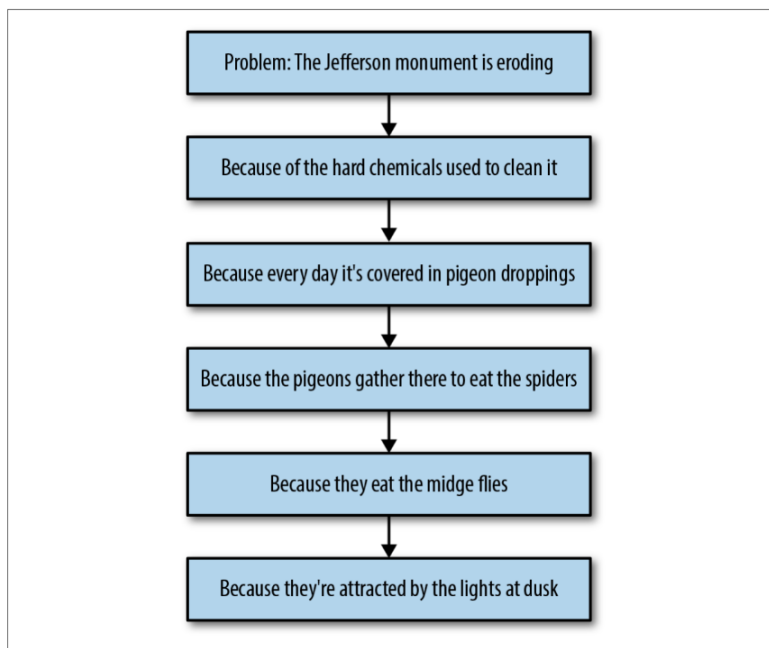
## Creating the Tree

To create the tree, you’ll first conduct a diagnostic analysis and then a solution-oriented analysis. These are separate exercises. It is tempting to jump to solutions without taking the time to gain a clear understanding of the true problem.

Represent your thinking in two separate trees. You may be familiar with the Five Whys, or fishbone diagram, which is also called an **Ishikawa diagram**. We'll use a similar structure to create the trees.

Once you are presented with a problem, ask why that would be the case. You may quickly see several possible reasons. Write each reason as the second level of the fishbone diagram. Then ask in turn why that would be the case, and write the reasons at the next level. Do this using the MECE technique (see **"MECE" on page 29**). Repeat a total of five times to come up with a set of possible root causes. Now you can make a declarative statement in your Ghost Deck (see **"Ghost Deck" on page 253**) that this is the problem and this is the root cause. For more on Ghost Decks, see **Chapter 9**.

My colleague at Sabre, Justin Ricketts, likes to use the example in **Figure 2-1** to help teams see how to approach a Five Whys analysis. It shows a memorable way of demonstrating how you can come to simple solutions and processes.



*Figure 2-1. Example of Five Whys*

So it turns out that the Jefferson Monument was eroding because the lights that illuminate the statue at dusk attracted tiny midge flies,

which attracted spiders, which attracted pigeons, which required cleaning crews to use harsh chemical processes to continuously keep it clean.

The solution to the problem was to simply turn on the lights that illuminate the statue one hour later—saving work crews, preserving the statue, saving on electricity and bulb replacement costs, and disrupting fewer tourists who come to observe the statue—and it cost nothing and took no time to implement. Justin’s illustration is a great way of showing how you can get to the root cause, but also how the solution may be easier than you’d think.

After you have determined some problems your strategy can address, and then figured out their root causes, you can start to formulate plans for addressing them through a variety of lenses and with tools we’ll explore in subsequent chapters.

The other part of the strategy scoping is to consider solutions. To do this, start by imagining the ideal end state—that is, what the world looks like after the problem has been solved or no longer exists. Make that declarative statement in your Ghost Deck. Ask “how” that state could be realized by determining what the prior necessary condition would have to be to achieve it. Do this in five layers of depth, regressing closer to the current state that has the problem. This will help you plan the path forward.

## Problems Versus Opportunities

Here we’ve focused, for the sake of brevity and convenience, on problem solving. But if you focus only on problems, the best you can do is maintain the status quo. Therefore, don’t forget to focus on *opportunities* for your strategy, things that represent gains to your customers and organization that they might not be aware they need.

This requires you imagining a better world, absent any direct feedback that people are hurting without it. For example, no one in 2007 was walking around the streets feeling the pain of not having a smartphone—they didn’t exist. No one had apps, and no one was sad about it. The iPhone didn’t directly address a clear and present pain that consumers felt at not having apps in their lives. But the invention represented a gain for consumers, augmenting and improving their lives and giving them conveniences they hadn’t thought of or knew they wanted.

Apple commonly employs this strategy of looking for customer gains, not just pains to solve. Take one of many other examples from the company: in 1998, no one was in despair or unable to be productive because their computers were only one color: boring black. But once Apple made the iMac in five colors named after fruit, the product sold like hotcakes, and is actually responsible for saving the company, bringing it out of the financial crisis created in Steve Job's absence. The strategy seems to have worked out OK for Apple.

## Hypothesis

*Let us employ the symbol 1, or unity, to represent the universe, and let us understand it as comprehending every conceivable class of objects whether actually existing or not, it being premised that the same individual may be found in more than one class, inasmuch as it may possess more than one quality in common with other individuals.*

—George Boole, *The Mathematical Analysis of Logic*

A *hypothesis* is a starting point for an investigation. When you hypothesize, you make a claim about why something might be the case, based on limited data, to offer an explanation or a path forward. You wouldn't make a proposition about something you are certain of. You may not have enough evidence yet to even convince you that it's true. But making such a claim puts a stake in the ground that suggests a path for focused analysis. In philosophy of logic, a proposition takes the basic form  $P \rightarrow Q$ , meaning "if P, then Q."

In your strategy work, there is no one single moment in which you declare a hypothesis. A hypothesis is a tool that gets worked into conversation, that gets used together with other tools as a helper. Unless you regularly keep company with strategy consultants, you won't often hear people say, "My hypothesis is..." (but strategy folks love the phrase). You have to recognize that when your team asks, "Why do you think this happened?" or "What's the reason for this?" you're being asked to state a hypothesis.

Consultants at Bain and McKinsey are hired at exorbitant rates to answer hard questions for CEOs. They might have an engagement to recommend whether the company should sell a certain division and exit the market or whether it should acquire a company, or they might be asked why profits are down in Europe, or what strategy the company should use to market in China. These are big, difficult,

strategic questions. If they were easy to answer, there would be no need for consultants.

These consultants will spend the next six weeks to six months answering these questions. They conduct research using every available channel, run workshops with key employees, and create recommendations. Their work product is a *deck*. These decks are usually very long and dense, containing loads of graphs and charts. This deck represents the answer to the key questions that started the engagement.

McKinsey consultants famously start engagements by quickly making a hypothesis, maybe after only a few days or hours on the job at a new company. Given that there is so much on the line, they don't work at the company, they may not have prior experience in the client's industry, and they may hold an MBA but be otherwise straight out of school, this sounds preposterous. But it isn't, and here's why. They're very good at forming hypotheses, using mental models similar to what we'll discuss here.

## The Five Questions

Hypothesis formulating is making a claim about the world: "this is that." Or, "the reason for X is Y." Or, "the way to make A better is to stop doing B and start doing C." I suppose you can just start making statements along these lines and call it a hypothesis. But that's not going to get anyone a strategy consulting job at McKinsey, and it's not going to serve you as a building block for your strategy.

This pattern is implied by the hypothesizing that strategy consultants do, but is not their process. So you might see very different material on this pattern in other sources. What I describe here I've adapted and customized based on my graduate studies in philosophy and what I have to put to work making successful strategies in my roles as CTO, CIO, and Chief Architect in a variety of companies.

Clinton Anderson was a Bain strategy consultant for 20 years. He once told me that his job in that time was about asking the right questions. The hard part is determining what the right questions are. Professor of Philosophy Alison Brown helped me see that in this context, hypothesizing (asking the right questions) tends to mean we start by asking these five key questions:

1. What is the conjunct of propositions that describe the problem?
2. What semantics characterize these propositions?
3. What are the possible outcomes?
4. What are the probabilities of each of these outcomes coming true?
5. What “ease and impact” scoring values suggest the right strategy?

This is our framework for asking those questions well. Let’s take them in order.

## 1. The Conjunct of Propositions Describing the Problem

When it’s time to perform an analysis, which is most of the time, we start with the first of our five questions: What is the conjunct of propositions that describe the problem?

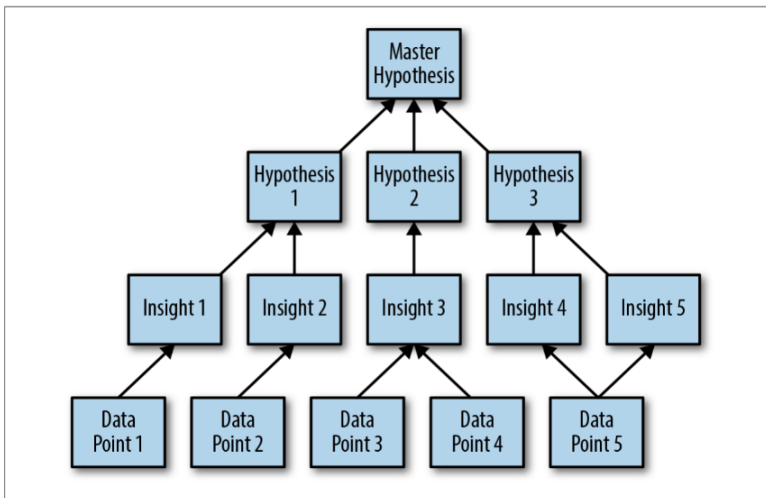
Twentieth-century philosopher Ludwig Wittgenstein was one of the leading thinkers in propositional logic. Propositions and propositional logic are well, but not definitively, explored in his book *Tractatus Logico-Philosophicus*, which I highly recommend. Ten years earlier, in 1911, Wittgenstein’s teacher Bertrand Russell wrote a paper titled “Le Réalisme Analytique” in which he describes propositions. Here we’ll unpack a few simple tools from this field to aid in our analysis.

In the *Tractatus*, Wittgenstein writes that “a proposition asserts the existence of a state of affairs” (section 4.21). So when you make a proposition, you are making a claim about the world. You are characterizing something that should be able to be expressed as a truth value.

When you are presented a problem, define it as a set of *propositions*. Each proposition is connected by the *conjunct* (the logical operator AND). Within each proposition, the *variables*, or constituent names, are also linked by logical connectors, so that you can deduce the truth value of the overall formula from determining the truth or falsity of each variable.

In modal logic, a proposition is true in accordance with its being borne out by the facts. So you must collect a few data points before making a proposition. Ultimately, your hypothesis will be a list of subhypotheses, each based on an insight, which in turn are each

based on a series of data points (see [Figure 2-2](#)). As we frequently hear from machine learning teams: if you think your data is clean, you haven't looked at it hard enough.



*Figure 2-2. Hierarchy of data, insights, and hypotheses*

Proposition P is a truth function of a set of constituent propositions if and only if you fix the truth value of P while determining the truth value of every proposition in the set. This is a cheerfully academic way of saying you have to be clear on what you are talking about: define your terms. People use “resource” to mean “compute power” or “human programmer.” When you say “customer,” do you mean the franchisor you are selling to or their customer? Your definition of “system” is likely too slippery to be talked about. So again, the simple solution is to define your terms.

## Insights

An *insight* is when you mix your creative and intellectual labor with a set of data points to create a point of view resulting in a useful assertion. You “see into” an object of inquiry to reveal important characteristics about its nature. In regular conversation, this is required all the time—for instance, to understand the punchlines of jokes. An old Groucho Marx joke goes like this: “This morning I shot an elephant in my pajamas. How he got into my pajamas, I’ll never know.” Getting the joke requires us to see into the ambiguity of language, that the word “in” has multiple meanings. It can mean

that Marx is wearing the pajamas, or that the elephant has found its way into Marx's pajamas, which we don't expect.

McKinsey publishes a set of its insights every year. This is a collection of conclusions and recommendations it's reached based on its surveys and independent research (the data points). You can read one at <https://mck.co/2MIY1Bs>. That document represents a rich set of examples of what we're talking about here. Let's take one example of forming an insight. The document states, "Culture is the most significant self-reported barrier to digital effectiveness." Then it presents a chart containing the top 10 factors technology executives cite as preventing them from effectively executing their digital strategies. This is not an insight, because it mixes no thought with the survey McKinsey conducted. It's just a representation of the raw findings—the data. Additional research from McKinsey indicates that several companies that have addressed their cultural problems head-on, by cutting down silos, have performed better and more quickly than competitors that have not. This is another data point. These data points are combined to reveal the insights that companies that are more willing to take risks, more responsive to customers, and more connected across diverse functions do better in the market. These insights lead to the hypothesis that executives must not ignore this fact and must not wait for their cultures to change organically, but instead must foreground and emphasize this specific kind of culture change—cutting down silos—in order to succeed at their digital strategy. *That's* making a claim, based on insights, based on data, and it recommends a course of action that is not obvious or intuitive. As you build your strategy, this is what you want to do.

Note that it's a good idea to read and cite material like McKinsey Insights reports in your Strategy Deck appendix, as part of your data point collection work, to help you reach your own insights that lead to your strategy.

Sometimes, to the untrained eye, a mere tautology can appear as an insight. A *tautology* is something that is *necessarily* true, so as to be redundant. That's not an insight. A tautology is an assertion—a proposition—that is true for every possible value of its constituent variables, so it's not useful. Nineteenth-century German philosopher Hegel refers to them as "trivially true": he's basically saying that although "A = A" is true, it reduces to making no claim, so it shouldn't be discussed like it matters—it's trivial. Sometimes people (ourselves even) speak in redundant or circular terms when trying



to define a problem. The statement “all bachelors are unmarried” is necessarily true as a proposition, but only because we have redundantly reworded the definition of bachelor: we have added nothing to our understanding. Watch out for tautologies as you perform analysis work in creating a hypothesis, or a Logic Tree (see “[Logic Tree](#)” on page 37), or in many other exercises in strategy creation where you need to form a real insight about the topic at hand.

## 2. The Semantics Characterizing These Propositions

Now let’s ask the second question in hypothesis formation: What are the semantics that characterize each of the propositions?

Here you are creating an interpretation, determining the discourse around each of these propositions. That’s because, again according to Wittgenstein, the “elementary proposition consists of names. It is a nexus, a concatenation of names” (*Tractatus*, section 4.22). But your interpretation must be clearly prescribed by a *domain of discourse*. To put it more simply, the word “play” means something different to a shortstop than it does to a theater goer than it does to a violinist than it does to a femme fatale in a film noir than it does to a toddler than it does to a deconstructionist philosopher. “Gradient” means something different to a data scientist than it does to a UI designer.

As you conduct your analysis, it’s powerful to realize that you are operating within a discourse, a *patois*, a learned and shared and, to some extent, private language. What are the terms you aren’t sure of? What are the terms someone else might not be sure about? Your work and the spheres of technology and architecture participate in what Wittgenstein called a *language game*. We use old words in new ways, and new words in old ways, and apply a word from one realm of life to another. Words have a preponderance of meaning.

This causes confusion, missed expectations, improper specifications, and incorrect application. It’s bad for software and organizations.

To sum up the point of this second of the five key questions: “stuff means stuff.” Being aware of the language games in which you and your teams are working is a great step toward being clear with your language.

This allows you to be clear on your definitions of each proposition, such that you can assign quantifiers and qualifiers with more rigor.