

O'REILLY®

Learning Systems Thinking

Essential Non-Linear Skills and Practices
for Software Professionals



Early
Release
RAW &
UNEDITED

Diana Montalion

O'REILLY®

Learning Systems Thinking

Essential Non-Linear Skills and Practices
for Software Professionals



Early
Release
RAW &
UNEDITED

Diana Montalion

Learning Systems Thinking

Essential Non-Linear Skills and Practices for Software Professionals

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

Diana Montalion



Beijing • Boston • Farnham • Sebastopol • Tokyo

Learning Systems Thinking

by Diana Montalion

Copyright © 2024 Mentrifx Group LLC. 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>). For more information, contact our corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com.

- Acquisitions Editor: David Michelson
- Development Editor: Shira Evans
- Production Editor: Clare Laylock
- Interior Designer: David Futato
- Cover Designer: Karen Montgomery
- Illustrator: Kate Dullea

- April 2024: First Edition

Revision History for the Early Release

- 2023-05-11: First Release
- 2023-08-24: Second Release
- 2023-11-10: Third Release
- 2024-01-04: Fourth Release

See <http://oreilly.com/catalog/errata.csp?isbn=9781098151331> for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Learning Systems Thinking*, 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.

978-1-098-15127-0

[LSI]

Brief Table of Contents (*Not Yet Final*)

Preface

Part 1: Introduction to Systems Thinking

Chapter 1: A System of Thinking

Chapter 2: Systems Thinking is Relationship Design

Chapter 3: Shifting Your Perspective

Part 2: You are a System of Thinking

Chapter 4: Self-Awareness as a Foundational Skill

Chapter 5: Replace Reacting with Responding

Chapter 6: A System of Learning

Part 3: We Are a System of Thinking

Chapter 7: Collective Systemic Reasoning

Chapter 8: Designing Feedback Loops

Chapter 9: Pattern Thinking

Part 4 (unavailable)

Chapter 11: Leadership is Synthesizing (unavailable)

Chapter 12: Embrace Uncertainty (unavailable)

Chapter 13: Redefine Success (unavailable)

Chapter 14: Find Places to Intervene (unavailable)

Preface

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the preface of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at sevans@oreilly.com.

“It became apparent that communications and computing served each other so intimately that they might actually become the same thing.”

—Tracy Kidder, *The Soul of a New Machine*

From Nowhere to Everywhere

Sixteen years ago, I owned an independent bookstore, Mooncougar Books, near the University of Montana in Missoula. The building once housed Freddy's Feed and Read, where you could nibble tofu Shepherd's Pie while browsing books by local authors. Eight years after Freddy's closed, I bought the business from the penultimate owner in a long line of struggling booksellers.

On July 20, 2007, twelve kids were nestled in the second-floor reading nook, wearing pajamas and reading Harry Potter and the Deathly Hallows. At midnight, the book was officially released and they left to continue reading it at home.

Downstairs on the purple counter, there were Read Banned Books buttons in a basket next to the bookmarks. The next morning, the enticing smell of Bears Brew coffee seeped in through the door shared with the coffee shop.

The bookstore, and Missoula itself, was surrounded by 1.6 million acres of National Forest land. During time off, I'd shrug on my backpack and disappear into that forest, hiking alongside fresh moose tracks. I rode my motorcycle (BMW F650, for the gearheads) across hundreds of miles of logging roads, sleeping in a tent overlooking the valley while a bear sniffed at my stuff.

Everywhere, indoors and out, I was reading.

One afternoon, I was avoiding boring inventory work, reading an essay by Anne Patchett published in (if memory serves) Real Simple magazine. Beauty, she said, depends on our geography. She wasn't beautiful in preppy Massachusetts towns. She wasn't beautiful in towns where black eyeliner and tattoos were common. As that article progressed towards the Big Reveal, the place where she felt truly beautiful, I knew exactly where we were going — the Bitterroot Valley. Where I lived. In the Bitterroots, being fresh-air tossed, friendly and smart, with some mud on your boots was beautiful. I rarely wore makeup, unless chapstick counts.

Who you were, authentically, mattered. In winter, getting up the mountain to home mattered. (Learning to put chains on your tires so you didn't slide off the mountain mattered.) In April, not getting stuck in mud mattered. Every shift in weather, especially during fire season, mattered. Bears mattered and moose occasionally meandered into my yard. My dogs played hide and seek with the deer. (I didn't know deer played hide and seek, did you?)

Reading and learning mattered. People said that Missoula had the highest per-capita percentage of waitresses with masters degrees. People mattered. When those waitresses asked you how you were, they actually wanted to know.

In Missoula, sixteen years ago, there wasn't technology. Not compared to, say, every town in the Northeast US. Before moving there, I'd studied programming and web engineering. In Montana, I had dial-up internet access. (Remember when we called the internet on the phone?) After a year or so of living down in the valley, I moved up a mountain and there, had no internet access at all.

Turns out, when you stretch a wire up the side of a mountain to provide phone service, it won't (later) carry the internet to you. Amazon's home page, such as it was then, took almost six minutes to load. Sure, I could check my email, if I was very patient, but basically I was cut off. I didn't mind. I had internet access in the bookstore. There was a receptor attached to the second floor roof, installed in winter, that stopped working when the leaves came back.

At home, I was surrounded by trees and wind and time to write. I did more writing then, though I didn't publish much. I was the source of my own entertainment. Books and words, mountains and rivers.

Life wasn't all peaceful and idyllic. There were dysfunctional interpersonal dramas, heartbreak, and difficult decisions to make. The relational part of my life was a mess. Still, when I

remember Missoula, I remember beauty. For many years, I planned to return when I could afford to stay.

(Missoula has broadband internet access now.)

Eventually, an internet service provider came up the mountain and sunk a tall metal pole into wet concrete near the hot tub. The installer stuck a receiver on top of the pole and pointed it towards a new transmitter. I bet you can guess what happened next

Within a month, I had moved into Second Life, a virtual world of endless ... enjoyment? Creativity? Time squandering? I rediscovered being plugged in. I say re-discovered because before Montana, there was Merentha, a text-based role-playing game my son and I spent too-many hours exploring together. One afternoon, he, as a Centaur, had taken me, a half-elf monk scholar, for a ride on his back to places where I could hunt dragons. I needed their skin to make sturdier gloves.

Missoula evenings became less about reading books; mornings became less about writing. I refilled my coffers from digital oceans. I surfed. Increasingly, I connected with people elsewhere. The rich beauty, the endless possibility, of Montana faded into the (boring) distance.

sigh

I don't know, writing this now, which decision was the chicken and which was the egg. After a local competitor refurbished and expanded their bookstore in some wonderful but impactful-to-me ways, I closed the store. I sold my inventory to Powell's in Portland, Oregon. Visits there still feel like visiting a loved one buried in a cemetery.

One winter weekend, in the midst of closing the bookselling business, I rented the Star Meadows Guard Cabin from the Forest Service in Whitefish, Montana. I'd recently finished a year-long, creative-nonfiction writing mentorship with Diana Hume George (which remains one of my most valuable investments.) Simultaneously, I was building custom software for cheap and practicing PHP, the programming language that, up until recently, powered most of the internet software. I brought the book *Six-Figure Freelancing*, determined to make a choice. Writing or programming? I was the only "techie" I knew in Missoula, which felt increasingly frustrating. The digital world was speeding up — I could hardly keep up from the middle of the forest.

From Software to Systems

Six months later, I moved to Austin, Texas. I'd never been to Austin but my Spreadsheet of Relevant Statistics told me that it was The Place to Be if you wanted to ride the wave of tech into the next decade. I choose to ride that wave.

Spoiler alert: it was the place. I rode the wave. And here I am ...

... living 90 minutes north of New York City, in a big house on four acres with my also-a-techie husband, three dogs, two cats, a ferret named Merry and eleven chickens. I've written code for smallish and medium-sized websites. I've written code for Really Big digital properties. Nowadays, I architect systems of interdependent software and cloud-native platforms for entire organizations.

I've helped build and design the digital traps that entangle your attention. I give talks and trainings at conferences, wearing those fancy headsets. I (sometimes) create healthy, happy teams that enjoy solving hard problems together. I pay more in annual income tax than I earned selling books in Montana.

When I first arrived in Austin, I built tools for clients like TXMPA, a nonprofit advocacy group that brings millions of film dollars into Texas. I spent a weekend, alone, moving lists of differently-structured data into their new CRM (people

information) software. The expertise I developed led to a meeting with a professional services team. I joined and we built big websites. I became deeply involved in open source.

Big websites, at the time, generally meant installing a piece of open-source software and extending it with (lots of) PHP. We built themes for styling, code that organized colors and fonts (we endlessly debated fonts). We hosted it all in a LAMP stack (software that runs the software that runs the software) with load balancing in front and controller/agent MySQL database configuration in back. Over time, we added hundreds of modules. To scale the software for high-traffic sites, we added caching (Varnish).

It was, often, complex and innovative work. Even though I could model the architecture of most features on a single page.

Analyzing a problem or feature request and then writing code to resolve or deliver it was enjoyable and addictive. Sometimes frightening, like when I added many lines of code to the Donate to Wikipedia process, enabling donors to pay with a credit card. Millions of dollars ran through that code. I threw up before launch but everything went fine. The internal code review from Tim Starling remains my favorite ever. Across multiple

initiatives and client teams I've worked with — the feedback I got from others has made the code stronger. Made me stronger.

Meanwhile, the world around us – the internet – was growing into the complex information graph it was perhaps never intended to be. Less about a document, or web page, and more about the relationship between all information. People were less interested in visiting a website and more interested in getting relevant information in whatever context they occupied, on whatever device they were using.

The digital gardens we planted grew into an interconnected web of interdependent information systems. I was with *The Economist* when they made the Big Jump towards a serious digital presence. I was with them again, ten years later, when a single article had 40+ destinations (website, app, Facebook, etc) and there were multiple types of media being created (including Films). New destinations and system-level challenges were arising daily despite no infrastructure (yet) to support them.

Nowadays, my teammates and I build information systems. I work across multiple teams building multiple platforms constructed from multiple cloud-native services interacting asynchronously with multiple types of software that

interdepend. The theme layer is now its own (decoupled) piece of software that morphs depending on the context (device, location, individual) it's serving. There is rarely only one frontend piece of software. The backends consist of multiple instances of editing or data-organizing software, in some cases, over 200 instances. In between, there are platforms, weaving software and services together with Kafka streams, infrastructure as code, container orchestration and data schemas defining what was once, simply, “content on a webpage”.

The complexity of everything has increased.

I am not alone. Most technology teams are attempting digital transformations, in one form or another.

Scaling now means “distribute information from multiple sources to nearly-infinite people, products and platforms.” To model this for you, I need interconnected pages of models. There’s a lot more caching.

The Coronavirus pandemic forced most organizations to rethink customer interactions, remap user journeys, and redesign data flows to give information significantly more visibility. Under the surface was a shift from serving requests

for web-based information to the need for asynchronous event-based interactions.

Television, like the internet, has shifted towards information platforms in your pocket. Technology is everywhere; everything is technology; everyone is involved. Technologists are learning to speak every language, tech, marketing, business, product simultaneously. We are continuously deploying, building data firehoses, moving from monolith to microservices, modernizing. Which involves transforming DevOps workflows with continuous deployment and integrating design systems into those workflows. We are adopting practices like Domain-driven Design and Team Topologies to help us create new mental models.

In other words, we are reconstructing the fabric of digital space time.

Here is where I almost burnt out and quit IT, imagining that life as an accountant or landscape architect would be preferable to systems architecture.

That transformation – from thinking in software to thinking in systems – is an iceberg almost every initiative hits.

Transformation initiatives don't just sink, they sink

spectacularly. And they drag down the hope and joy that powered them in the first place.

It's exhausting.

There is a lot of blame. There is a lot of drama. There's a lot of bullshit. This book is born out of the bullshit.

We've been amassing a pile of reasons for our failings since the “software crisis” began in the 1960s, as complexity was increasing. Then, since, and now, the reason is the same: *we can't change our thinking fast enough to keep up.*

The skills we need no longer fit the stereotypical Lone Supernerdy coding while eating pizza at 2am. **No one can do it alone.** Technology skills (alone) are not enough.

We don't think in systems. To move from software to systems – we need to think differently (together).

Technology Design is Communication Design

Once again, I have arrived at a crossroads in my life. Here I am ... writing. The choice I made in Montana, a career in

communication or a career in technology was not, and should never have been viewed as, a dichotomy. My journey from nowhere to everywhere, from software to systems, has led me to one inevitable conclusion: *technology design is communication design.*

“Organizations, who design systems, are constrained to produce designs which are copies of the communication structures of these organizations.”

—Conway’s Law

We build what we think. We structure thinking through communication and then we craft it into something actionable. Communication is how we think together. The structure and flow of information, in our own minds, with other people and as organizations creates the sociotechnical systems that are the core of our work.

Whatever else we think we are doing, we are always, also, designing systems. Wherever we imagined we were heading with our digital transformations, we have arrived. Information technology is no longer elsewhere. Systems are right here, woven into the fabric of our days, our lives. They influence our decisions and shape our mental models.

We are immersed in the digital world, like I once was in Second Life. We have all traveled away from a friendly conversation with the waitress and into a global, always on, digital pageant.

My technology career has long since surpassed any dream I had when I left Montana. When I'm quiet and honest, I also know that I went deep into the disease as well, like a drunk who isn't entirely sorry for everything that happened before she got sober. But, mostly, would have preferred making choices without all the delusion. Mistakes have also been my teachers on a journey that has led me here: advocating for topics that aren't, necessarily, popular at tech conferences. (Where's the Kubernetes chapter?!) Yet, my day-to-day experience suggests they are essential IT topics.

We ignore them at our peril.

I entered my STEM career through the linear, logical door of programming internet software. I abandoned the messy creative uncertainty of nature and words for the solidity of cities and software engineering. Then I found myself in the hall of mirrors that is systems.

This is the risk everyone in STEM, perhaps everyone in life, takes. We choose a path and end up right back in the middle of

the forest. Because everything is a system. Everywhere we are part of those systems.

We need to think in and communicate about systems.

Part I. Introduction to Systems Thinking

Chapter 1. A System of Thinking

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 1st chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at sevans@oreilly.com.

“Vision without systems thinking ends up painting lovely pictures of the future with no deep understanding of the forces that must be mastered to move from here to there.”

—Peter Senge, *The Fifth Discipline*

You might imagine that when you finish reading, *Learning Systems Thinking*, you will have learned to think in systems.

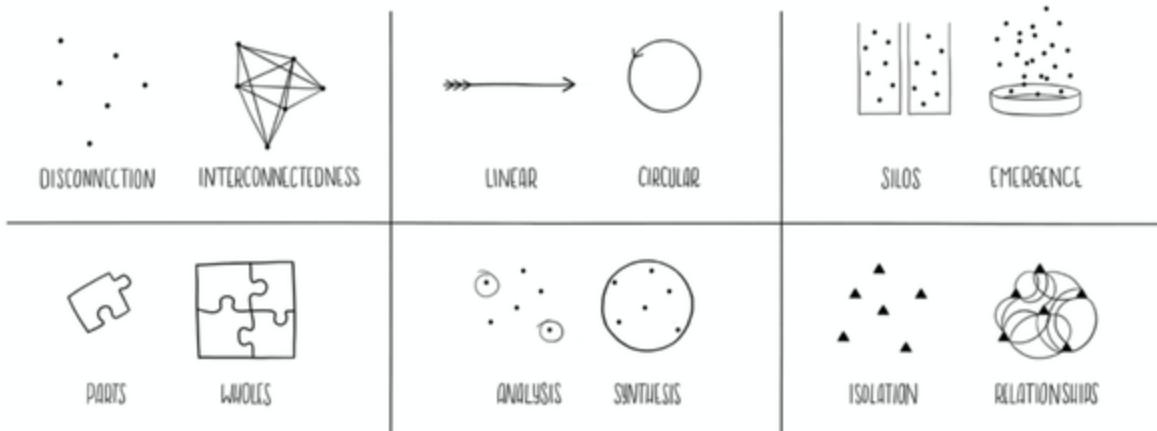
Nothing could be further from the truth. Systems thinking is a practice, a perspective, a framework, an emerging language ... we could even call it a way of life.

Reading a book about tennis doesn't teach you to play tennis. You must go outside and play tennis. Experience is needed to change your thinking. I hope that while you are reading this book, you will go outside and play with systems.

This book describes a system for thinking about systems. Reading it will give you context, guidance, vocabulary and practices. What does a world in which we practice "systems thinking" look like? How do I navigate towards it? What do I do there? Why does it matter? What practices, principles and tools will I need to be successful?

My favorite quick introduction to systems thinking is this model (figure 1-1).

TOOLS OF A SYSTEM THINKER



Leyla Acaroglu - <https://www.disruptdesign.co/>



Figure 1-1. Tools of a Systems Thinker. Credit: emmasegal.co, illustrator

For technology professionals, this is a journey from thinking about software to thinking about systems of software. It is also, perhaps more importantly, a journey away from thinking about people and technology as separate entities. What we think, and communicate, is what we push to production.

Software approaches arose in the machine age and we cling tightly to reductionist and mechanistic approaches. 40 years ago, our systems were primarily physical and mechanical. We read the printed newspaper, books from the library, letters in the mailbox. When we got lost, we asked a human for directions. The type of thinking we needed, and were taught, in some ways fit our time.

Now we are in the systems age. We've created a vast, interrelated and interdependent digital landscape of software to replace newspapers and libraries and postal mail and maps. Software shapes our online spaces, like stores and social spots. Software structures infrastructure for other software. Relational complexity, the number of variables we need to consider in order to make software decisions, is running riot. As software becomes systems of software, we have reached the limits of our traditional ways of thinking.

"In the Systems Age we tend to look at things as part of larger wholes rather than as wholes to be taken apart."

—Russell L. Ackoff

This book is unashamedly about thinking. About abstract thinking. Many, many, many, many times in my career, I have heard derision towards thinking (and communication) as matterful skills and activities. “That’s too abstract. Make it concrete!”

I’m all for concrete. My house is built on a concrete foundation. But my stove and my bathtub and lawn and garden and dogs are not concrete. Concrete thinking is important but it isn’t *everything*. Creating concepts that guide our integrated

decisions making, synthesize multiple points of view, take us deeper into a problem to discover its root cause, and transform the mental models that structure our legacy software systems ... those things matter too.

In the wilderness of systems, there is a lot less concrete.

Here we will expand our toolset as knowledge workers, rather than stay stuck in the constant, pointless, internal culture war between architecture and engineering as a practice. Systems are nondualistic – the choice isn't “ivory tower architect” vs “hands-on coder” ... the choice is “which tool, practice or approach will help us understand what we need to understand?” How do we shift perspective and think differently?

The first step towards systems thinking is: Willingness. Willingness to see things differently, to think deeply and practice self awareness, to become curious about patterns and complexity. Willingness to change your mind and step into the unknown.

As Morpheus says to Neo in The Matrix:

“I can only show you the door. You're the one who has to walk through it.”

Let's walk through it together.

In This Chapter

You will learn that systems thinking is a system of thinking about systems. The book will introduce foundational thinking practices that, when done together, improve our nonlinear thinking skills. The concepts we will explore include:

- Linear thinking is the default structure of “thinking”.
- Systems thinking is nonlinear.
- “Systems thinking” has a myriad of definitions that focus on how “the sum is greater than the parts”.
- Systems thinkers develop qualities that, over time, improve their capacity for thinking about complex situations.
- Systems thinking needs to be practiced by an individual, group and organization.
- Systems thinking is a form of leadership that shifts the paradigm (a typical pattern) from command and control to creating systems of support for integrative thinking and decision making. Alas, you will be riding “on the front of the train”.

The practices you'll discover are:

- Reading and learning about systems thinking.
- Begin noticing your own thinking patterns and the circumstances around you.

Linear Thinking is the Default

“... black-and-white logic can sometimes work even for complex systems. But it ignores the ways in which parts interact with one another. Reductionism may serve to explain how a bird flies, but not how a flock of birds move in unison. It may describe internal combustion, but not traffic patterns. It may describe electric patterns in the brain, but not consciousness, and it’s unlikely that anyone or anything – not even the world’s most powerful computers – will ever fully analyze the interactions that make for healthy soil.”

—Mark Bittman, *Animal Vegetable Junk*

We are taught to think linearly. Linear thinking is so ubiquitous, many of us don't recognize it as one type of thinking. We call it, simply, thinking: predictable, rational, repeatable, procedural, dualistic, top down, and concerned with control. We rely on linear thinking to design, build and deploy, run and maintain software.

Our experience in software has strengthened our linear thinking skills. Governed by our “if this, then that” causal thinking, we expect software systems to behave exactly as we intend them to behave, in all circumstances. In code, we speak a language designed to be unambiguous. We expect the people who build them to behave in predictable, procedural, top-down controlled ways. Our preferred communication style reflects these expectations – straightforward, concrete and concerned with control.

Linear thinking is reductionistic, understanding a whole by breaking it into parts. Object-oriented programming is reductionism. We rely on software architecture approaches that divide and conquer, “manage” complexity through modularity, “decompose” problems into subproblems. We break software (or a system of software) into parts or components. We model boxes with lines between them. We fit people (teams) into those boxes.

When we outgrow that model, we recreate the boxes.

We can’t operate as knowledge workers without linear thinking. In some circumstances, excellent linear approaches are exactly what are needed. Linear thinking and approaches help us to:

Break down a complex problem into its component parts so you can understand and solve it.

Analyze cause and effect. Find the “bug” causing an unplanned effect. Design a change to produce a desirable effect.

Imagine the steps involved in building something new and take those steps, adapting as you learn more.

Learn new programming languages, tools, frameworks, rules and processes. Apply those skills.

Identify weak ideas and implementations then work to improve them.

Iteratively build new software behaviors, learning from the result.

Identify and follow best practices. Change them as circumstances change.

Test ideas to discover where they fail. Track valuable operational data.

Improve efficiency by editing solutions until they are elegantly simple.

This book does not argue against linear thinking. Linear thinking isn’t bad, and systems thinking isn’t good, like the

Witches of the East and North in the Wizard of Oz. Linear and nonlinear thinking are interrelated. This book is a matterful addition to your bookshelf because linear approaches cannot resolve systemic issues.

We are in the systems era. As relational complexity increases, we need to think *differently*. Many of our challenges are systemic. We need to expand our skillset so we can think, communicate and act as healthy systems.

Systems Thinking is Nonlinear

“Linear relationships are easy to think about: the more the merrier. Linear equations are solvable, which makes them suitable for textbooks. Linear systems have an important modular virtue: you can take them apart and put them together again— the pieces add up.

Nonlinear systems generally cannot be solved and cannot be added together. . . . Nonlinearity means that the act of playing the game has a way of changing the rules. . . . That twisted changeability makes nonlinearity hard to calculate, but it also creates rich kinds of behavior that never occur in linear systems.”

—James Gleick, *Chaos: Making a New Science*

In a linear world, I would plant seven kale seeds in my garden and 50 to 55 days later, I would harvest seven mature kale plants for salad. When a rabbit nibbles one of my plants, I put a fence around the garden to keep it out. When the days become unusually hot, growth will take longer, so I adjust my expectations. Perhaps next year I plant the seeds earlier (or later) in the season. This is the mindset we use when planning software initiatives.

In my actual garden, I sometimes get nine plants because kale is biennial and I planted some last year. Sometimes I get zero plants. When that happens, maybe it's rabbits, maybe it's deer reaching over the fence because their favorite foods are in low supply. Maybe there's been too much rain or not enough rain or too much heat in May or October. Maybe the soil doesn't have enough nitrogen or the cabbage worms or slugs ate them. Maybe the birds poked holes in the leaves while eating the snails eating the kale.

Most likely, the cause of zero plants is *some combination of these things*, impacting each other in difficult-to-predict ways. This, in its simplest form, is what we mean by nonlinear. Systems are not fully controllable and unpredictable. Relationships between dynamic parts impact what happens, not simply the parts themselves.

The Biggest Little Farm is a documentary about Apricot Lane Farms, started by John and Molly Chester. They designed a well-balanced ecosystem and developed rich soils that produce nutrient-dense foods while treating the environment and the animals with respect. In other words, the Chesters used systems thinking to create a reawakening agricultural ecosystem from a dying farm.



Figure 1-2. <https://m.imdb.com/title/tt8969332/mediaviewer/rm3577464577/>

Like the Chesters of Apricot farm, technologists can apply a systems thinking mindset when they design software initiatives. Working with the actual world, whatever that may be, instead of trying to shape the world to fit a linear, traditional approach.

You may have built a single piece of software that, for the most part, operated in a relatively-controlled environment. When I did that, I couldn't always predict what would happen when I pushed a change to production. But for the most part, I could mitigate the risk of unexpected outcomes.

For many of us, myself included, that time is past. Software is everywhere, built on top of other pieces of software, interacting with multiple information sources, spanning the breadth of an organization with hacky bridges. New features are used in ways

developers didn't not intend, Modern software becomes "legacy" three minutes after we launch it. Software operates in ever-changing circumstances and depends on a flow of information in flux.

Some bad news: Nonlinear approaches are invariably more difficult than linear ones. Your life is not about to get easier. Does this make you want to throw the book across the room? You already have so much to do. And nonlinear approaches involve doing the work **and** figuring out how to do the work **and** improving the ways we work **and** clarifying why the work matters **and** learning ... all the time.

Learning all the time. Thinking deeply and also thinking deeply about thinking.

The good news is: You will become more *effective*. The work won't feel so much like work, which means you'll have more energy. We aren't trying to make everything *easier*, are we? As knowledge workers, *we are trying to improve our capacity for doing difficult things*. Nonlinear approaches increase signal and decrease noise.

We are doing difficult things. In the world of information systems, we've built an entire digital ecosystem in less than 30

years. Software governs most of our human communication processes. That ecosystem is emergent ... the sum is greater than the parts. Behaviors and trends arise from the relationships between the parts. The “like” button didn’t just give Facebook users a quick way to comment, it transformed social constructs in unpredictable ways.

We can’t design nonlinear systems with linear thinking. For that, we need to expand our thinking toolset.

What is Systems Thinking?

“For those who stake their identity on the role of omniscient conqueror, the uncertainty exposed by systems thinking is hard to take. If you can’t understand, predict, and control, what is there to do?”

—[Donella Meadows](#)

Nonlinear thinking is expressed in a myriad of forms. This book is called *Learning Systems Thinking*. That phrase, “systems thinking”, has been defined in numerous, sometimes contradictory, ways across technology, business and academia. The vocabulary, which I will explore in the next chapter, is still emerging. Nonlinear thinking integrates more than systems

thinking. Strategic thinking, for example, is systems thinking plus creative navigation towards change. Pattern thinking, parallel thinking and systemic reasoning ... all mean thinking in systems.

Learning teams are nonlinear thinking teams.

To some extent, it doesn't matter too much whether or not we share an unequivocal definition. The practices and the vocabulary we use to describe them are evolving. Our definitions will get us started. It's okay if they morph over time.

To define systems thinking, let's first consider what "system" means. Here are some definitions:

- A set of things working together as parts of a mechanism or an interconnecting network (Oxford)
- A regularly interacting or interdependent group of items forming a unified whole (Miriam-Webster)
- A system is a whole that consists of parts, each of which can affect its behavior and properties. The parts are interdependent. (Russel Ackoff)
- An interconnected set of elements that is coherently organized in a way that achieves something (a goal).
(Donella Meadows)

- An arrangement of parts or elements that together exhibit behavior or meaning that the individual constituents do not. (The International Council on Systems Engineering)

Another definition of system is: a set of principles or procedures according to which something is done; an organized framework or method. This is important because systems are sociotechnical. The principles and procedures people use and the framework or methods that structure our work ... will inherently design our systems. However else we define “system”, it includes the ways we structure our thinking and approach.

As software professionals, we are concerned with *software* systems. So for our purposes, I’ll define system as: a group of interrelated hardware or software elements that interact and/or interdepend to serve a shared purpose.

Caution - this clear description gets muddy fast. Defining the “purpose” depends on your point of view. Every piece of software serves a purpose. Wordpress is a digital publishing tool. But that’s not what I mean by purpose. Purpose is a property of the whole and not in any component. What mission do the elements serve?

For example, imagine a small business whose mission is to tell the world about the health benefits of plant-based cooking. Wordpress might be an element in that system. Other parts like social media publishing tools and Google Ads and an asset manager and recipe-generation platforms might also play a role. The writer's labor in developing the information is an element in the system. As are the readers. Without the inputs (content) and outputs (consumers), the software doesn't serve a purpose.

Thinking has a surprising variety of definitions. When we are systems thinking, we use our minds to reason about something. Which is an interesting sentence ... if I am using my mind to think, who is the me using my mind? We aren't going to open a philosophical conundrum, delightful as that might be, but I do want to make the point that system thinking is not only thoughts but also your awareness of thoughts. Thoughts that are intertwined with experiences, memories, judgment and sensory information (like the look on someone's face).

In this book, we extend the definition of thinking to include structuring and communicating those thoughts. Whiteboarding, to understand something, is thinking. We don't simply think about code, we write code, we don't simply think about systems, we make artifacts. An artifact can be a document or a model, a

Slack message or a conversation, anything that conveys thinking. In this way, systems thinking is hands on.

What, then, is systems + thinking? Here are some “systems thinking” definitions:

- Systems thinking often involves moving from observing events or data, to identifying patterns of behavior overtime, to surfacing the underlying structures that drive those events and patterns.¹
- Systems thinking is a holistic approach to analysis that focuses on the way that a system’s constituent parts interrelate and how systems work over time and within the context of larger systems. The systems thinking approach contrasts with traditional analysis, which studies systems by breaking them down into their separate elements.²
- A systems thinking approach means recognizing that a sum is greater than its parts — that all the pieces of an organization connect, interact and play a part in outcomes.³
- The definition of systems thinking states that it is a cohesive approach that views all subsystems as parts of an overall system, rather than in isolation or as segments.⁴
- [Systems thinking] recognizes and prioritizes the understanding of linkages, relationships, interactions and interdependencies among the components of a system that

give rise to the system's observed behavior. Systems thinking is a philosophical frame, and it can also be considered a method with its own tools.⁵

NOTE

In this book, I am defining systems thinking as a Practice, a system of foundational thinking practices that, when done together, improve nonlinear thinking skills.

Systems Thinking is a Practice

The practices in this book don't represent the Definitive List of All Necessary Practices. Systems and nonlinear thinking are evolving, especially in the world of software systems where we are just beginning to apply it. These practices are an excellent starting point, you will discover more on your journey.

How we categorize the practices you'll find here doesn't really matter. For example, they blend "hard skills" and "soft skills" (in my experience, the soft skills are harder). Practicing will expand the thinking tools in your toolbox. Over time, practicing will improve your ability to apply these tools to wicked problems (problems with many interdependent factors that are in flux).

Systems thinking is a practice that happens in your own mind and between people. It restructures the thinking and communication processes around you, like the organizational structure at work. Practicing systems thinking inside of a linear-thinking organization is helpful but not transformational. This book will introduce practices to support thinking alone, with others and as a leadership practice.

The four parts of this book, and the chapters within each, are designed to introduce you to the capabilities of a systems thinker. In [Figure 1-3](#), you'll see the capabilities introduced in Part 1:

Capabilities of a Systems Thinker

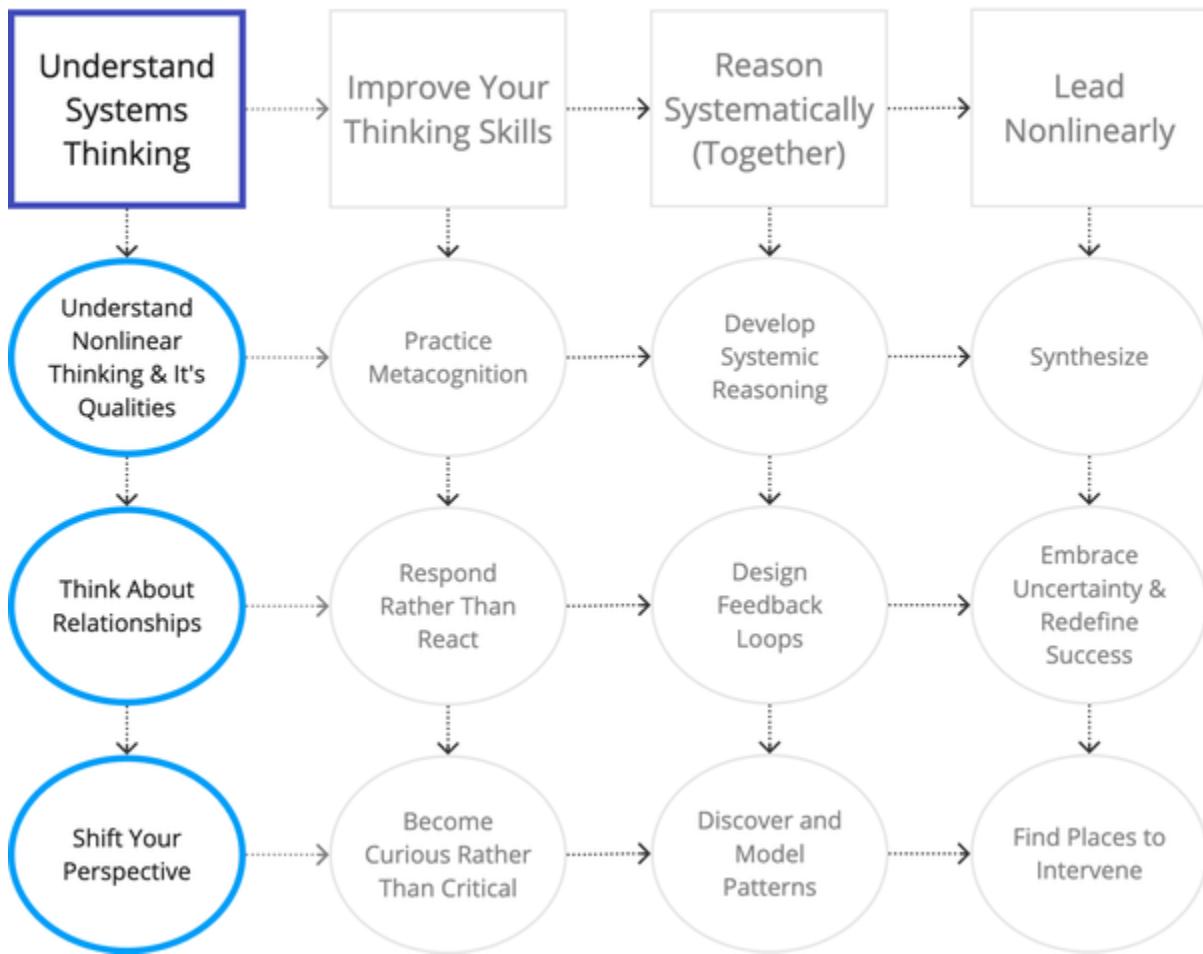


Figure 1-3. The capabilities of a systems thinker, part one highlighted

In Part 1, you are learning the basic concepts and practices. The three subsequent parts of this book each represent an essential area of focus:

- Part 2: Working with your own mind.
- Part 3: Working with input from other people's minds and the system itself
- Part 4: Leadership in a nonlinear world.

Your Own Mind

NOTE

You cannot improve your thinking if you are not aware of your thinking.

Each of us is a system. Systems thinking begins and ends with self awareness: becoming aware of your own thinking patterns and improving the quality of your thinking. Practices we'll explore in this book, like writing and deep work, learning to manage your reactions and proactively designing your own learning will develop your system thinking qualities. The healthier your individual thinking system is, the more positive impact you can have on the systems around you.

Systems are the default reality. When we lack awareness of this reality, we potentially generate more harm than help. Becoming systems wise can improve the quality of your life beyond your technology career. In [Figure 1-4](#): you'll see the capabilities introduced in Part 2.

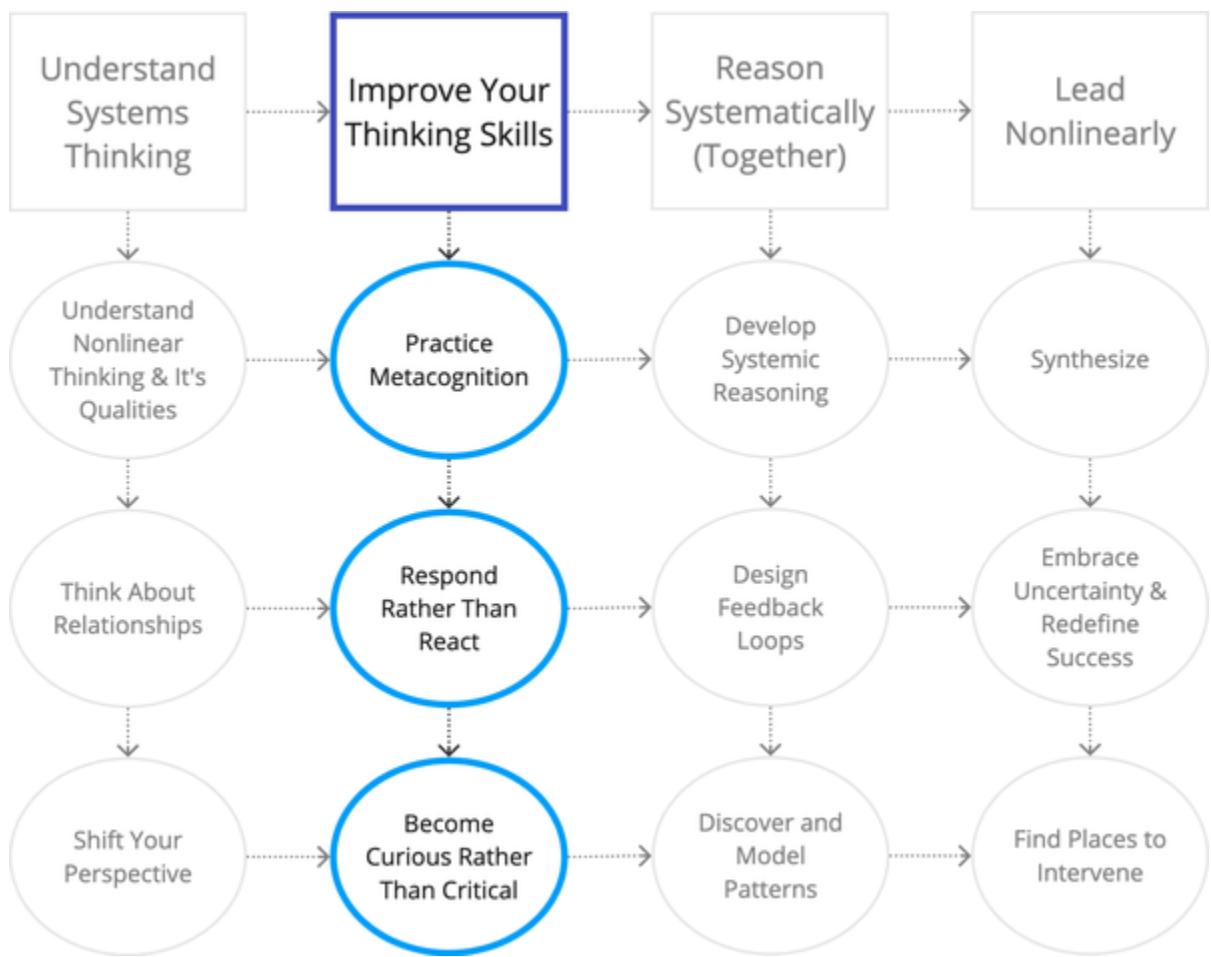


Figure 1-4. The capabilities of a systems thinker, part two highlighted

Thinking Relationally

Unless you work alone on a desert island, building software nobody will ever use, you are part of a community. The groups you engage with form systems of thought and behavior that, implicitly and explicitly, design the solutions you push into production. You can improve the quality of team-level thinking (and behavior) with practices like:

- Building sound recommendations using systemic reasoning and strengthening your reasons for acting.
- Crafting conceptual integrity through feedback loops.
- Understanding (and designing) relationship patterns.

In [Figure 1-5](#): you'll see the capabilities introduced in Part 3.

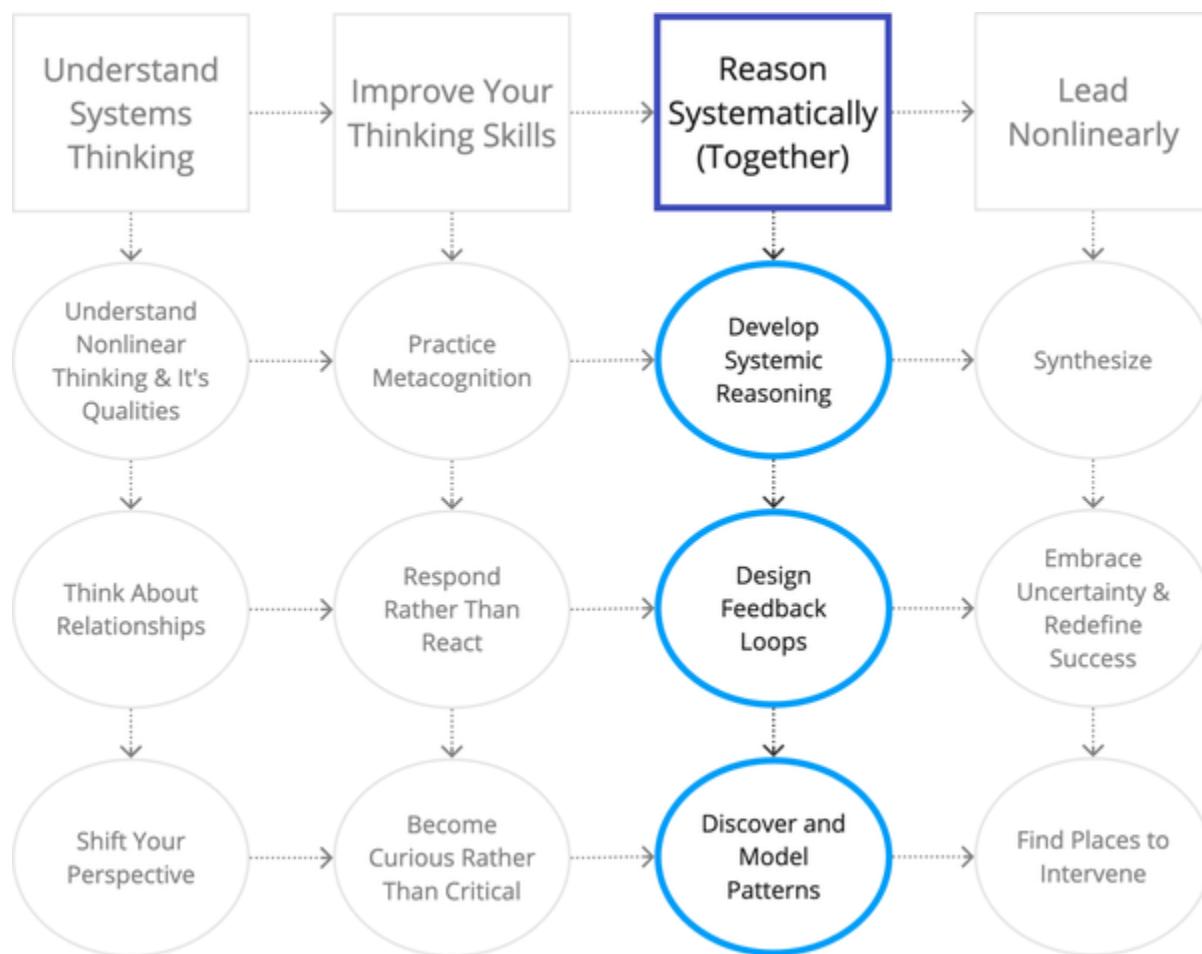


Figure 1-5. The capabilities of a systems thinker; part three highlighted

Leadership in a Nonlinear World

Our daily-life communities form bigger systems; organizations, industries, countries, ecosystems, and our one planet among billions. When we work on a software system, the boundary of that system usually lies outside of our team. The mental models, structures, feedback loops and patterns of everyone involved or impacted creates that system and defines its purpose. We provide systems leadership whenever we:

- Synthesize knowledge, experience and sound judgment (using collective modeling).
- Embrace uncertainty and redefine success (shift from linear to nonlinear expectations).
- Identify root causes and places to intervene.
- Create frameworks for knowledge sharing that encourage system thinking approaches.

In [Figure 1-6](#): you'll see the capabilities introduced in Part 4.

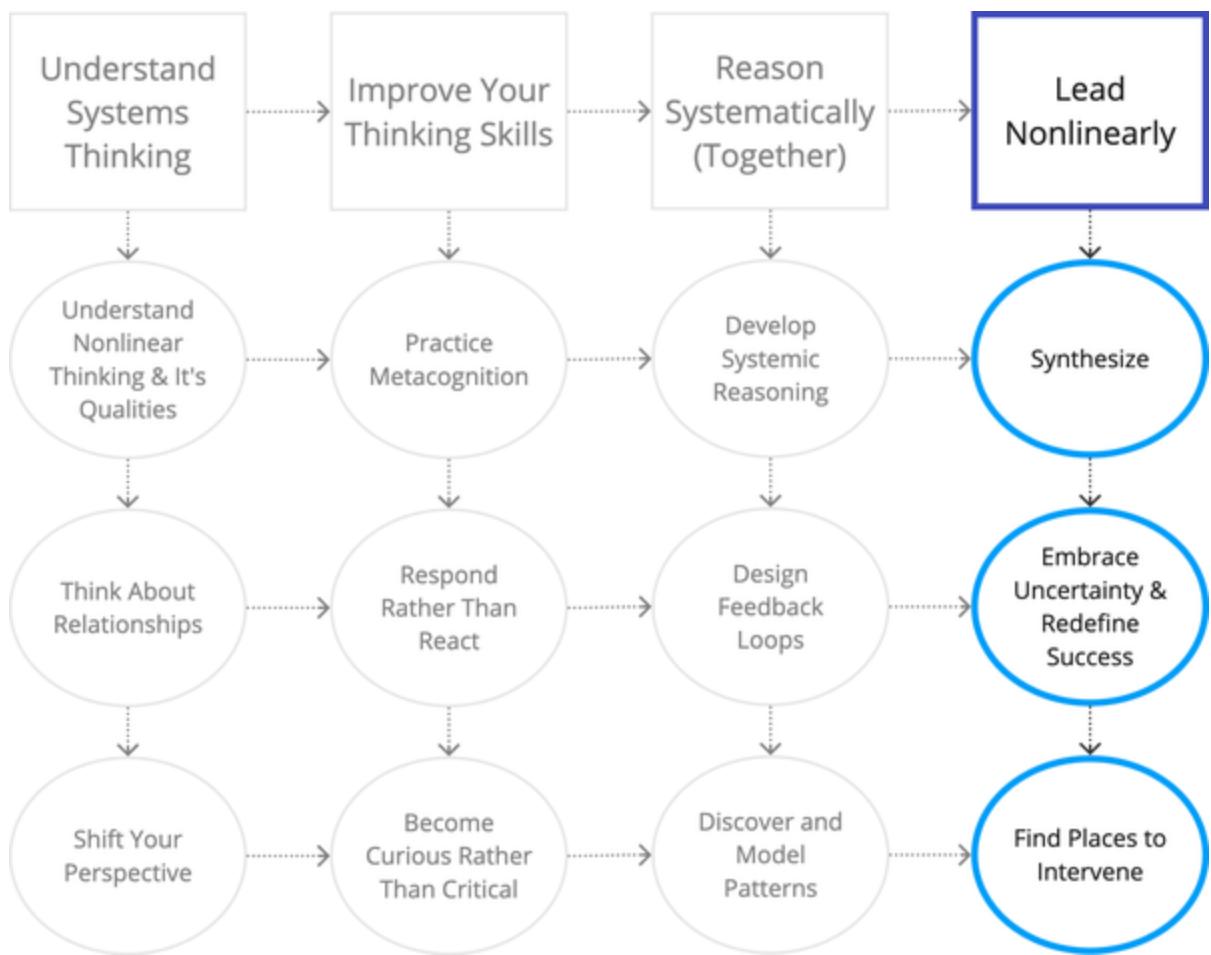


Figure 1-6. The capabilities of a systems thinker, part four highlighted

Taken together, these practices are potentially powerful because they are emergent ... meaning that you will experience benefits because you practice them together that you can't get when practicing one alone. It is the relationships between the parts, always and forever in systems, that generate transformation.

These practices, over time, enable us to prioritize the qualities and capabilities found in a systems thinker.

Qualities of a Systems Thinker

There are no “12 Steps to Systems Thinking and the First One Will Surprise You!” There are no systems-thinking whiteboard test to pass or certification exams. You never finish learning about systems, they will teach you forever. So how do you know you are becoming good at systems thinking?

As you practice systems thinking skills, you will take on more-complex challenges. With each new challenge, you will discover there is much more to learn. The paradox in systems thinking is that the more you know, the more you know that you don’t know. We could say that the most valuable quality of a systems thinker is they know they don’t know.

Here are some recognizable patterns and behaviors in people who think nonlinearly. People who are good at systems thinking

...

<p>Recognize the difference between reductionistic, analytical thinking (linear) and taking a systemic perspective (nonlinear). Discern when to apply one or the other (or both).</p>	<p>Create conceptual models, alone and with others, to guide impactful decisions. Have sufficiently-diverse modeling techniques and use them improvisationally.</p>	<p>Practice thinking. Demonstrate high-levels of self awareness and metacognition, especially about thinking patterns that are reactive, fallacious or biased.</p>
--	---	---

<p>Understand how interrelated and interdependent parts act together to create patterns and processes. Investigate the ways those patterns are reinforced.</p>	<p>When faced with recurring problems, seek to understand the systemic structures and feedback loops that block change.</p>	<p>Know that people systems are inextricable from technical systems. View challenges as inherently sociotechnical.</p>
---	--	--

<p>Communicate well-</p>	<p>Shift perspective</p>	<p>Approach life</p>
---------------------------------	---------------------------------	-----------------------------

reasoned ideas, recommendations and theories. Articulate the reasoning behind conclusions. **Listen respectfully** and helpfully work with others to strengthen collective insights.

easily to explore challenges from different points of view. Comfortably engage shifting mental models as circumstances change.

with an “always learning” mindset. Structure discovery, learning and exploration with others proactively.

Accept that uncertainty is a natural, welcome and inevitable part of life.	Use systems vocabulary to talk about change.	Describe solutions that are meaningfully connected to the context and system-level purpose.
---	---	---

As a systems architect, I do systems thinking full time, but you don't need to take that path. You can develop systems skills to compliment your software-thinking skills. A complimentary career ladder for most software professionals might look something like [Figure 1-7](#):

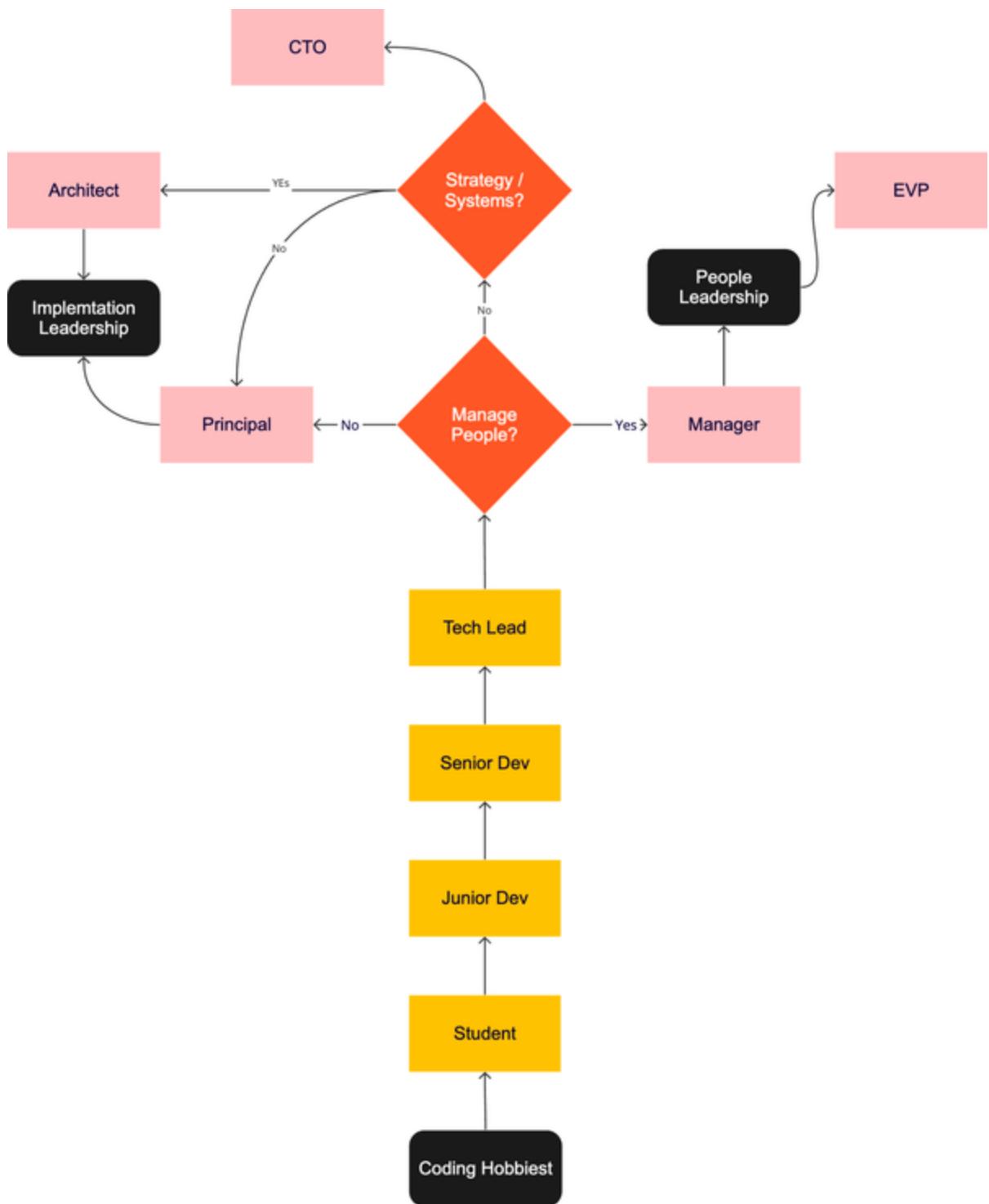


Figure 1-7. A career ladder for systems thinkers

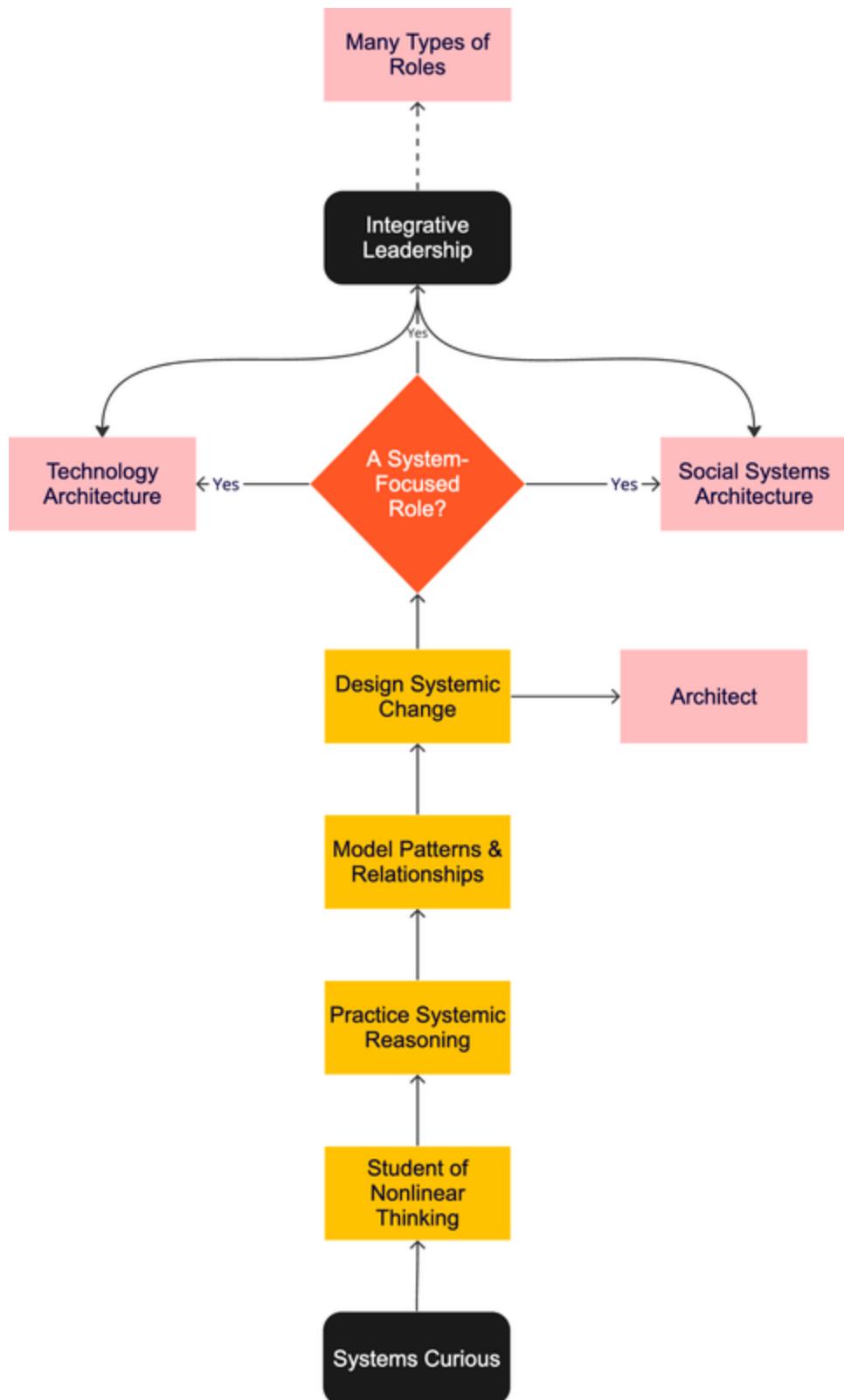


Figure 1-8. A career ladder for systems thinkers

Systems thinking is a mindshift, away from linear and reductionist approaches towards nonlinear ones. The “right” way to achieve this mindshift depends on your skills and circumstances. There are many right ways.

If you find that ambiguity troubling, buckle up, because systems thinking is full of ambiguity. You won’t be careening around blind corners recklessly though. You are shifting your expectations and enjoying the journey without always needing to know exactly where you are going.

Chances are, you are already shifting. If you’ve adopted microservices or continuous integration or other interrelated software approaches, you’ve experienced some of this mindshift. If you do collective modeling, connect what you build to why it’s valuable, or think about the relationships between software parts, you are thinking about systems. If you’ve improved patterns and processes in order to “fix” a recurring problem – systems thinking.

When software systems are changing, software delivery skills like Go or AWS implementation tools or hypermedia API design are valuable and necessary. But they do not predict success. Brilliant software developers, product managers and strategic

leaders are often sincere and hard working, yet making little progress. We are all blocked by two obstacles:

1. We are spectacularly terrible at nonlinear thinking. We are constantly tangled up in our opinions, cognitive biases, fears, assumptions, conditioning and logical fallacies. It takes practice to skillfully and consistently untangle ourselves.
2. We don't know that we are terrible at it. In fact, the worse we are at nonlinear thinking, the more certain we are that we are good at it!

Therein lies a paradox: We must be good at nonlinear thinking in order to see that we aren't good at nonlinear thinking.

Fortunately, as Carl Jung said:

“Only the paradox comes anywhere near to comprehending the fullness of life.”

Systems Thinking as Leadership

Throughout this book, I will describe systems thinking as a type of leadership. Systems leadership is integrative, combining ideas and experiences with other ideas (discarding some along

the way) to form a cohesive whole. The goal of integrative leadership is conceptual integrity, which we will define in the next chapter.

You can provide systems leadership regardless of your role. As Carol Hanisch said, “the personal is political”. Demonstrating well-developed self awareness, emotional intelligence and empathy can have a powerful, positive impact on the circumstances around you. Constantly-improving your listening, thinking and communication skills is, in systems, a prerequisite for sound leadership.

If linear leadership is ensuring people follow a recipe, nonlinear leadership is making the best meal together from available ingredients. It is understanding what “best” means under the circumstances.

Quality systems thinking can not happen in a vacuum. We need systems of support around us. Whenever you are in a position to do so, championing systemic reasoning and other practices you’ll learn in this book will help structure this support.

In Part 4, we will explore this type of leadership more deeply. Meanwhile, think about, as you read, the impact of systems thinking practices on organizations as a whole. Look for

opportunities to shift the thinking. Here are some indicators that system thinking has taken root in an organization, compared to indicators that very little systems thinking is happening:

High levels of systems thinking support	Low levels of systems thinking support
--	---

Collaborative	Siloed and hierarchical
---------------	-------------------------

Strong rhythm	Chaotic, interrupted rhythm
---------------	-----------------------------

Why is this valuable?	What are we building?
-----------------------	-----------------------

Respectful discourse	Bullying (subtle or overt)
----------------------	----------------------------

Trust building is evident	Lack of trust is evident
---------------------------	--------------------------

“Best possible solution under the circumstances”	“Meets requirements (or not)”
--	-------------------------------

Dedicated to meaningful work	Dedicated to managing work
------------------------------	----------------------------

Transparent	Information is a power currency
-------------	---------------------------------

Better-than-expected solutions are normal	Blame is normal
---	-----------------

Strong agreement on matterful best practices	Constant in-fighting
--	----------------------

Pattern aware	Pattern repetitive
---------------	--------------------

High levels of systems thinking support	Low levels of systems thinking support
Forgiving	Grudging
Lead by influence and knowledge integration	Lead by command and control
Present and engaged	Absent yet maintains authority
Truth and insight seeking	Constantly shifting goals and values
Self awareness	Everyone blames “leadership” or other teams
Emotional intelligence	Derision
Proactive	Reactive
Ask questions, in front of the team	Give orders, in an email or private channel
Well-prepared for meetings, lots of whiteboarding	Most people are silent in meetings, collaborative visuals are rare
Empathy and respect	“Resources”

High levels of systems thinking support	Low levels of systems thinking support
Videos on (usually)	Videos off (usually)
Enjoyable	Exhausting

Systems leaders proactively navigates towards the most valuable destinations, while paying attention to impact. They create well-reasoned, insight-driven and well-considered recommendations. When the status quo is pushing things in the wrong direction, systems leaders don't mind standing alone (for awhile).

Riding on the Front of the Train

Practicing systems thinking and nonlinear approaches has welcomed benefits. You can tackle more-complex problems, recommend changes that make a big difference, grow and learn constantly, build things that matter to the people who use them, be inspired by others and sometimes share inspiring insights. You will be correct more often (in the long run) and, more importantly, you will enjoy thinking with others.

There is one challenge that is, perhaps, not as welcomed. Sometimes you will be alone, thinking differently, with no one

(yet) validating your perspective. In the face of invalidation, sometimes you will change your mind because your thinking is unsound. Sometimes you'll stand your ground because your thinking is sound. Always, it can be challenging to discern the difference yourself.

I have struggled with this challenge, both personally and professionally.

“Nobody thinks like you, Diana.” I am standing in the kitchen, leaning on a table edge, across from my partner (at the time) who is sitting on the counter. He is ranting about a work situation, full of pent up frustration. I’ve just offered a strategic recommendation for changing his situation. He didn’t welcome it.

Bitch Don’t Fix, I call it. Blaming our frustration on external circumstances but not changing those circumstances. That’s what he’s doing, most days. He is certainly not alone, I Bitch But Don’t Fix. You probably do too, at least sometimes. There is, as we will see later in this book, a lot of blame in systems.

“Nobody thinks like you.” He rejected my thinking because it didn’t fit “the norm” in his situation. Ironic, given that the norm in his situation was frustrating him. I was thinking differently.

Perhaps you've had this experience? If you haven't, as you improve your ability to think in systems, you will. Having a perspective that is "outside the norm", feeling alone with a reality denied is, I'm sorry to tell you so early in our journey together, part of systems design. An architect colleague calls it "the front of the train."

My colleague, Mark, first used that phrase during a meeting in London. We were facing each other, sitting on uncomfortable chairs in a borrowed office. I was jetlagged and Bitching Not Fixing a situation I'd flown over to discuss. "They just don't get it!" I said. "This is important!", I said.

"Diana," he replied gently but firmly, "you are on the front of the train. You look out and see a forest. You say, 'Look at the trees!' People riding in the other cars say, 'What are you talking about? That's a lake!' You don't get to be mad at them. They aren't looking at the trees yet. Riding in the front of the train is your job."

Front of the train is systems thinking and design. Which brings us straight to the hard cheese: There is no technology you can adopt, group you can join, role you can be promoted into, tool you can use or solution you can recommend that will work universally. This book won't tell you how to "fix" situations

using management techniques, Kubernetes clusters or hiring more “juniors”.

There is no magic bullet. Even if there was a magic bullet, you’d be standing there holding it, and nobody would believe you. You need to show them, over time, how the magic works.

Nonlinear approaches do not value being right. Being right is always temporary. Instead, nonlinear approaches value seeing things as they are (and can be). Getting input that strengthens our thinking. Gracefully letting go of wrong views. And being patient, kind and curious about other people’s perspectives while we learn (and learn and learn) to be “right” together.

We need other people’s insights. What works in any circumstance... depends. Depends on the context, depends on the people, depends on how integrous (or not) the communication flow is. Depends on what you say and why people don’t want to hear it. Depends on a myriad of factors.

Systems thinking is figuring out what “it depends” on. Then communicating new ways of seeing until people can see it.

How to Get the Most From This Book

Each chapter will narratively explore a topic, recommend practices and offer a list of supportive resources. You'll find recipes for mixing and matching practices. As you'll see in Chapter 3, systems thinking is the process of transforming our mental models in order to transform our systems. You'll need all levels of practice to deliver real change.

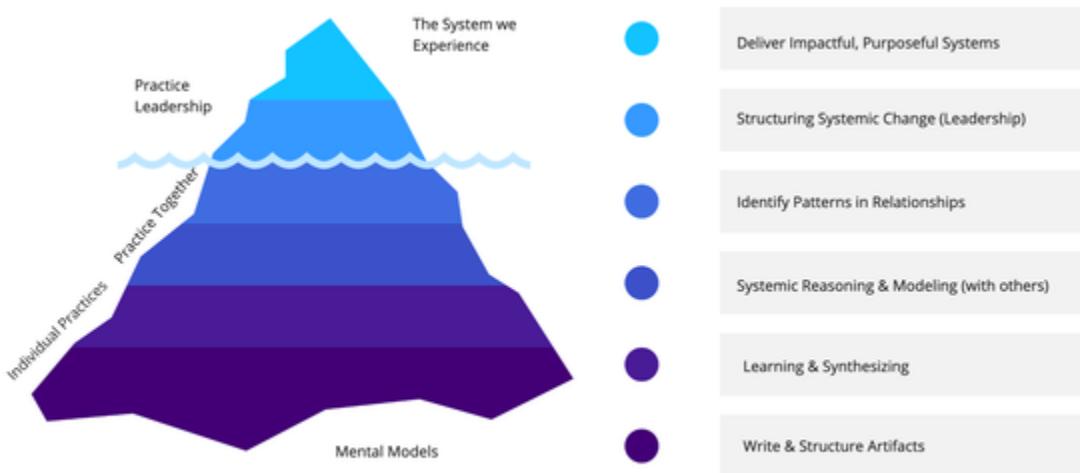


Figure 1-9. Caption to come

I will encourage awareness, learning, modeling, exploring, thinking and experimenting. More importantly, you'll write.

“If you’re thinking without writing, you only think you’re thinking.”

—Leslie Lamport, distributed systems scientist

One of the best ways to create conceptual integrity, to think better, is to write. You can also model and code and even express your ideas through interpretive dance – whatever works for you. But you will struggle to improve your thinking if you don't also write.

I'll encourage you to make artifacts, because we test our thinking through communication. Artifacts are any intellectually creative act: emails, Slack messages, documents, UML, morse code, domain models, stories, Miro boards. Artifacts generate feedback loops and feedback loops are the science of systems.

Code is an artifact. So is infrastructure. Prototyping is an impactful way to share our thinking and I highly encourage it. In my experience, combining words, images and working code is a strong way to share systems thinking.

Practice. I'll say that word, again and again. There is no whiteboard test at the end of this book. There is only you engaging with your thinking, other people's thinking, the circumstances you are in, and the patterns reinforcing those circumstances.

You'll try things ... and see what happens.

Summary

For most of us, linear thinking is the default definition of “thinking”. The qualities we value are predictable, rational, repeatable, procedural, dualistic, top down, and concerned with control. These qualities help us build software. But as we move further into the systems age, away from mechanistic processes, linear approaches don’t scale.

Systems thinking is nonlinear. It focuses on the sum of the parts, the relationships and patterns that make up the whole. Other forms of thinking, like pattern thinking, are nonlinear. Nonlinear thinking is a system of practices that improve your capacity for thinking (together) about complex situations and acting in ways that improve the system as a whole.

We practice systems thinking, first and foremost, in our own minds. We practice with others. Ideally, regardless of our role, we also provide organizational leadership that nourishes and supports systems thinking.

This book will teach you many of those practices – but the practices themselves (and the systems you consider) are the real teacher. Practice will help you shift your mental paradigm from linear, command and control approaches to integrative

approaches. These words might not mean much now but by the end of the book, you will have systems thinking tools in your toolbox, ready to use in daily life.

Practices

Read and Learn

Like all things systems, you'll benefit most from experiencing multiple points of view. Here are some systems thinking, or nonlinear, resources to get you started.

- *Thinking in Systems*, book by Donella Meadows
- [Leverage Points: Places to Intervene in a System](#)
- *The Fifth Discipline: The Art & Practice of the Learning Organization*, book by Peter Senge
- Scale?
- Dr. Russell Ackoff on [Systems Thinking](#)
- [A Lifetime of Systems Thinking](#) by Russell Ackoff

Keep a journal

Throughout this book, I'll encourage you to write down your thinking. Writing is thinking and a core practice for engaging your thoughts (and the thoughts of others).

Each chapter will have questions to explore in your journal. Most chapters will include additional prompts. Begin now by finding a notebook and pen to keep with you while you read. You can use a digital device instead but, as you'll see in Part 2, I'll strongly encourage you to write by hand (sometimes).

Prompt: Notice, and note, any challenges that are reoccurring, especially the ones where solutions applied haven't worked or have made the situation worse. (Those challenges will be good candidates for systems thinking practices in later chapters.)

Questions for Your Journal

- How would you define “systems” or nonlinear thinking? Describe how it’s necessary for a software professional.
- Describe 3 qualities you already have as a systems thinker and at least one you don’t have but want to cultivate.
- When you are having a “front of the train” experience, what helps you navigate? What can others do to support you?
- What are two qualities of systems leadership that you think would benefit your organization most? Why?

· <https://thesystemsthinker.com/systems-thinking-what-why-when-and-how/>

- | <https://www.techtarget.com/searchcio/definition/systems-thinking>
- | <https://www.phoenix.edu/blog/what-is-systems-thinking.html>
- | <https://study.com/learn/lesson/systems-thinking-approach-model.html>
- | <https://ahpsr.who.int/what-we-do/thematic-areas-of-focus/systems-thinking>

Chapter 2. Systems Thinking is Relationship Design

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 2nd chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at sevans@oreilly.com.

“You think that because you understand “one” that you must therefore understand “two” because one and one make two. But you forget that you must also understand “and”.

—Donella Meadows, *Thinking in Systems: A Primer*

Donella Meadows defines systems thinking as how “*parts together produce an effect that is different from the effect of each part on its own*”.

How relationships produce effects

Systems design is relationship design. Software becomes a system of software when “parts together” achieve something that could not exist without the “together” part. When “parts together” no longer fit our circumstances, we can’t simply change the software. We need to change the way software interrelates.

When we think in systems, we think about relationships between interdependent and interrelated elements. Elements are anything that impacts how a system behaves.

When I say that systems thinking is relationship design, I do not mean solely between software parts. I mean the human parts too, our thinking, behaviors, mental models and communication patterns. I mean the whole ball of wax, not just the code running in production. I mean maintaining interconnection between the code, the world around it and the system’s purpose in that world.

Let's use a neighborhood as an example. We can model elements in a neighborhood with a map showing property boundaries containing structures housing X number of people or providing a service (like a police station or elementary school). Will this model help us *understand* the neighborhood? Will it help us predict how the neighborhood will change over time as circumstances change?

In a neighborhood, homes, infrastructure like water and sewer, ecosystem aspects like wildlife or locally-grown food, economic trends and many other factors are in relationship to one another. People are part of that relationship but so are the demographics they have in common (or not). These relationships form circumstances, events change those circumstances in unpredictable ways.

A sudden event, like the closing of a factory or a slower change process, like gentrification, trigger systemic processes with complex causes and no single solution. Interdependent changes transform a neighborhood in unique ways. Changes can be restrictive, like the loss of water access or expansive, like improvements in digital connectivity. A change in traffic patterns turns a quiet area into a noisy one. The global pandemic impacts neighborhoods designed for commuters. None of these changes are shown on the property model. Events

transform a neighborhood *because of the relationships between elements.*

When our software systems don't behave as we intended, we consider that a failure. I find this relentlessly surprising because unexpected results are far more common than "everything worked perfectly on the first try." Why not embrace the fact that everything we try, if we pay attention, can teach us. We just need to expand our view.

When you think about systems, you consider things like:

- How do the parts interdepend?
- How do they share information?
- What are the patterns that keep the relationships functioning?
- What are the patterns that block the system from evolving?
- Can we improve the relationships and patterns to deliver the highest-value outcomes?
- How do the circumstances change over time?
- What happens in response to a high-impact event?
- What is the root cause of that event?
- Within these circumstances, what are the structures that support (or prohibit) recurring behaviors?

- When considering a specific action, does it improve the system's ability to serve its primary purpose?

In This Chapter

You will learn that systems thinking is relationship design, though not simply between software parts. Systems leadership is looking for root causes in the mental models that structure our thinking. The concepts we will explore include:

- Systems thinking is concerned with how relationships produce effect.
- Conceptual integrity is the most important consideration in systems design.
- Our strategies for systems are often making things worse because of counterintuitiveness.
- Systemic relationships are dynamic, always in flux.
- Systems are sociotechnical, there is no hard boundary between people processes and technology. They are interrelated.
- Time is always a factor in systemic relationships.

The practices you'll discover are:

Free writing

We will explore writing practices more deeply in Chapter 4. At the end of this chapter, there are warm-up prompts.

Modeling as a tool for systems thinking

Using the basic systems model as a template, modeling a process you've experienced and identifying any ways it might be improved.

Creating Conceptual Integrity

Our ideas design our systems. When our ideas are cohesive and in good relationship to each other; when they are supported by healthy, shared patterns and principles; when we push code changes that improve the system's ability to serve its purpose, we create conceptual integrity.

Conceptual integrity must exist between people before it can exist within a technology system. Creating cohesion in our thinking process creates cohesion in our technology systems. This is a chicken and egg assertion, which comes first? Our thinking and the physical systems, over time, come to resemble each other.

“Conceptual integrity is the most important consideration in system design.”

— *Frederick P. Brooks Jr., The Mythical Man-Month: Essays on Software Engineering*

Our systems can be diverse and interdependent while still having a wholeness that grows and adapts as things change. When a software system lacks conceptual integrity, it contains “many good but independent and uncoordinated ideas” (Brooks). Does that sound familiar? It describes every software system I’ve seen.

A metaphorical example: One part of the organization wants a car. Another part wants a boat. So the engineers are told to build a carboat, which nobody wants. Conceptual integrity helps us integrate points of view and design changes that matter most to the system as a whole.

Conceptual integrity is tricky to define but we know when it’s missing:

- We can’t share data across the digital ecosystem because of silos. (In the technology system and the people system.)
- Software is wired directly to other software with that one python script someone wrote ten years ago. We can’t break

that chain, ever.

- We can't work effectively across teams or roles. Teams openly distrust (or even dislike) each other and defend silos. Patronizing communication is considered normal.
- We can't "do DevOps" because the legacy software has become a giant ball of mud with painful weekly (monthly, quarterly, yearly) releases.
- Product teams avoid building things "the right way" because it's too hard. They've built seventeen "side products" that are duct-taped together. Three of them are doing the exact same thing for different teams.
- There is no shared, semantically-meaningful data structure between the software. One database talks directly to another database, skipping the logic (software) layer because, well, that was the easiest thing to do.
- There is no domain language in the software, it's difficult to tell what the parts are there to do.
- I could go on and on and on

Technical debt rises when our solutions lack cohesion. Thinking well together across the many individual moving parts, creating conceptual integrity, is fundamental to thinking in systems.

Counterintuitiveness

Creating conceptual integrity can be complex. Our thinking about systems is (usually) linear. We think “If I do X then Y will happen.” Perhaps. More often though, that is not how systems work. Sometimes you change one thing and all hell breaks loose. Sometimes you give users the feature they most want and nobody uses it. We can’t be certain how a change, in the midst of relational complexity, will play out.

A service that returns critical information is too slow. The team adds autoscaling to make the responses faster under load. It works as intended until the 3rd party software that the service uses to translate timezone data stops responding. The interdependent software’s database couldn’t return asynchronous results fast enough. The critical information isn’t merely slow now, it’s dead until a fix is applied.

Systems are counterintuitive, meaning that the “right” answer to a systems challenge will rarely be the one fix that our linear minds offer. We blame the technology for what are actually people problems. We fail to intentionally design a system then blame the architecture for holding us back. We make something go faster, then flood downstream systems.

We have developed an intuition for familiar systemic patterns. To solve systems challenges, we usually need to change those

patterns. We are blind to, and uncomfortable with, ideas that run counter to our “intuition.” When we are thinking in systems, the best answer will often push against what we “know”.

Jay Forrester, system pioneer at MIT, describes how many organizations know exactly where their systemic blocker is, where they need to make a change.

“Then I’ve gone to the company and discovered that there’s already a lot of attention to that point. Everyone is trying very hard to push it IN THE WRONG DIRECTION!”

In the oft-quoted *Mythical Man-Month*, Brooks expresses what is, perhaps, the most famous example of counterintuitiveness in tech:

“Adding manpower to a late software project, makes it later.”

There are always knock-on effects and blindspots. Systems thinking helps us proactively look for them. Actions that lead us towards our goal are *always* an experiment. We increase our chances of getting the desired result when our initial recommendations are sound, relevant and cohesive. As you’ll

discover in Chapter 7, we practice systemic reasoning in order to create sound and cohesive recommendations.

Counterintuitiveness isn't a bad thing – it is inescapable. We will go the wrong way, make a bad situation worse, fix a mistake with another mistake. In systems, we aren't trying to do the right thing every time because that is impossible. Instead, we are always trying to learn.

A Basic System: Everything in Flux

Through experience, the good stuff and the frustratingly-recurring stuff, we learn. As we learn, we evolve our concepts. Conceptual integrity isn't a static thing, it's a quality constantly created in a system, nourished by our decisions like healthy soil.

We design software to operate in a particular circumstance, at a particular time, serving a particular purpose. But things change. Software interrelates with an ever-changing world. Which means we are ever-changing our conceptual models. For example, when we move from monolith to microservices, we need to think differently. Not because of all the implementation differences, though those are significant. The monolith fit the

world it was born into. The microservices are needed for a new world.

A systems perspective allows us to ask “what changed? How does the system no longer serve its purpose? (And if the system is serving its purpose just fine, why are you redesigning it?) When paradigms shift, we don’t simply need new technology tools – we need new patterns and relationships (in the tech and in the people building it). Systems thinking helps us see this.

In Donella Meadow’s simplest drawing of a system in figure 2-1, we see the fundamental parts of all systems: inflows, outflows, the current state of the system. We also see the conceptual parts: our goal for the system and our analysis of progress towards that goal.

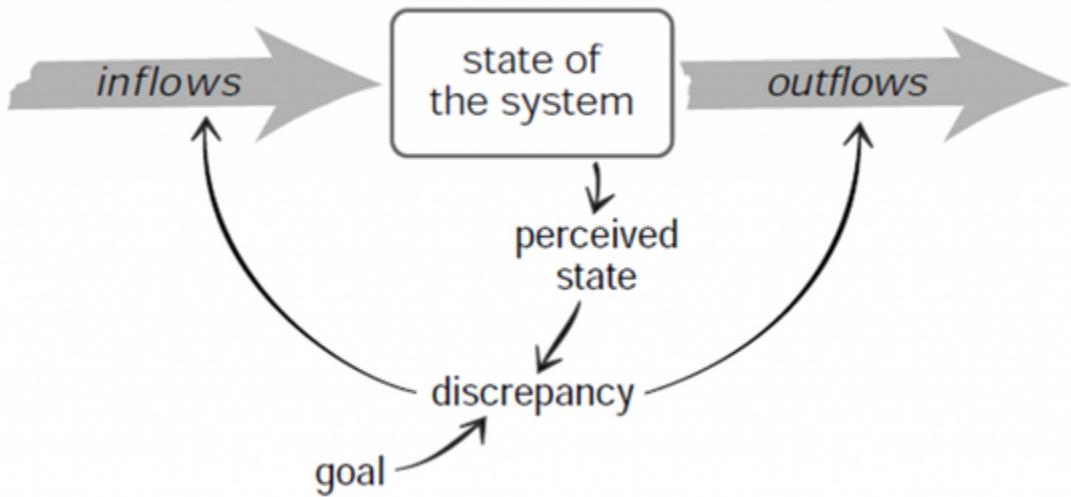


Figure 2-1. Donella Meadow's basic system model

The model's "state of the system" box can hold tangible things, like water, money or heat. For example, the current state of "temperature in my study" is 72 degrees Fahrenheit. I feel chilly (perceived state) so I decide to raise the temperature to 76 degrees (goal). I increase the hot air flow into the room (inflow), while keeping the windows closed so little heat escapes (outflow).

At first, the discrepancy between the current state and my goal is 4 degrees. Over time, the discrepancy becomes zero. *Time is always a factor* in systems because the state changes over time.

This simple model consciously hides the complexity in most real-life systems. If you only have one goal at a time, it is a linear-thinking model. This almost never happens in real life. Also, you can't control everything happening outside the state box while the discrepancy diminishes.

In my chilly office, I can put on a sweater. My goal is to be warmer, raising the room temperature is only one way to do that. I can raise the temperature and put on a sweater, what would that process look like over time? I can use a fireplace, which raises the temperature in the room unevenly. A snowstorm rolls in, lowering the temperature outside and drawing heat from the room faster as I try to heat it.

The model is incredibly useful to help us think in systems and develop a shared language for systemic problem solving. But it's essential to keep in mind that it does not represent a linear process – everything is in flux. It is simply a construct that helps us think well together about systemic goals and how we might alter patterns to achieve them.

A System of Ideas

The state box can hold intangible things too, like trust or empowerment. For example, lying about the status of work in

progress lowers trust (over time) and transparency increases it.

To understand systems thinking, imagine that the state box is full of ideas. Ideas about how the system works, how it should work, what needs to be changed and how people should work together. Ideas about top priorities, methods for delivering software, OKRs and which capabilities are business-critical are in there. The box includes everyone's opinion about which tools are best to use, how to ensure quality, and what defines a good culture.

In figure 2-2, I've added thinking:

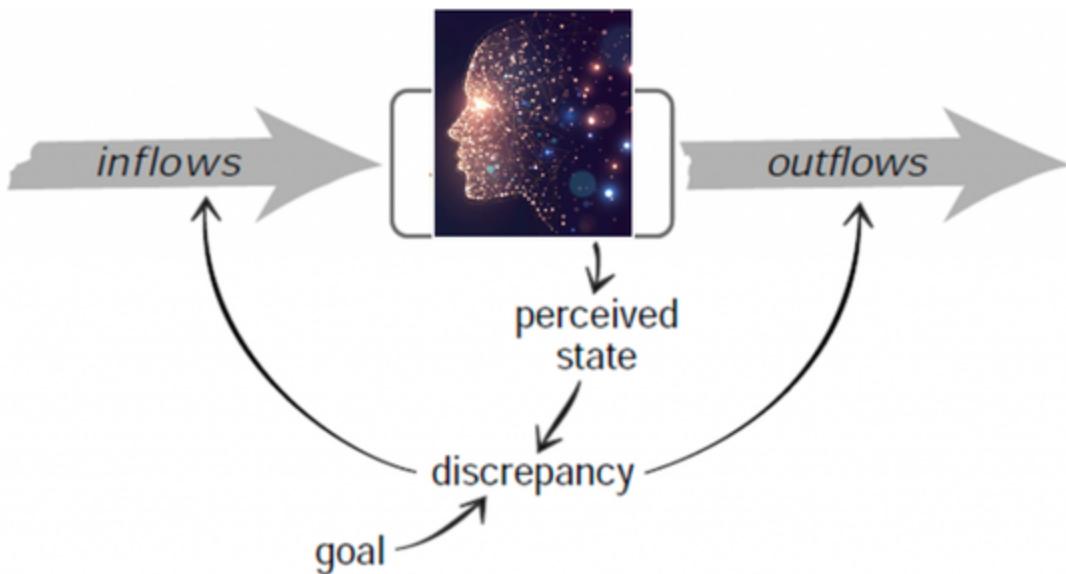


Figure 2-2. Donella Meadow's basic system model plus thinking

Thoughts flow in and actions flow out. In the state box, there are a myriad of attitudes, points of view, opinions, recommendations, knowledge about the technology system (both accurate and inaccurate) and its impact on people ... all tumbling around and becoming entangled.

Systems thinking, as a whole, is the art and science of setting goals and designing processes that decrease discrepancies, over time ... in circumstances where there are always multiple, competing goals happening simultaneously and constantly changing circumstances around the model. For example, how many technology initiatives got scrapped when the pandemic hit? How many Suddenly Business Critical initiatives kicked off? How much work in progress was suddenly, irrevocably disrupted?

When we think in systems, we create cohesion, harmony, a sense of order from that pile of idea laundry. We integrate and synthesize disparate ideas, knowledge, points of view and activities until we can see what matters most to do, under the circumstances. We look at the patterns and relationships, improve our understanding of how things work, until we can see a path towards matterful change.

Then, we keep looking, learning, creating cohesion, because everything is always in flux. We have imagined a world that wasn't in flux, was controllable, but if that world ever existed, it is now becoming extinct.

Systems Thinking is Sociotechnical

“So, what is a system? A system is a set of things—people, cells, molecules, or whatever—interconnected in such a way that they produce their own pattern of behavior over time.”

—Donnella Meadows

There is a big mindshift for many software professionals that is also the most important one: Software systems are sociotechnical. Three microservices running in Google Cloud are a collection of software. Collections are not systems. What makes a collection a system is the relationship between parts. Those relationships include the human parts and the interactions between them.

Humans think about and decide what to build. Ideas are rewarded, punished or ignored based on shared organizational structures, mental models and the social feedback loops. Linear thinking doesn't simply define how we think about coding,

software and infrastructure. It shapes what we expect from other people – predictable, rational, repeatable, procedural, dualistic and concerned with control. We structure a top-down approach to thinking. We expect our social systems to share these qualities, even when those expectations block us. Reality often challenges our social structures. Demanding that something be delivered in three days, for example, doesn't make it possible.

“Agile was invented because reality refuses to bow down to power.”

—Joe Eaton, PhD, systems engineer

Initiatives like “digital transformation” or “modernization” or “monolith to microservices” aren’t (strictly) top-down initiatives because they aren’t a linear change. If an organization tries to change the structure of their software system without changing the structure of how people think together, they will likely fail to deliver innovation.

“The true system, the real system, is our present construction of systematic thought itself, rationality itself, and if a factory is torn down but the rationality which produced it is left standing, then that rationality will simply produce another factory.”

—Robert Pirsig

Systems are sociotechnical, the people processes and behaviors shape the technology-oriented processes and behaviors. And vice versa. We inhabit, you could say we embody, the systems we develop. Small changes in our thinking at the individual level scale into improved decisions, patterns and behaviors in our broader circumstances. Everyone benefits when individuals and teams become better at thinking about nonlinear, asynchronous changes.

Software serves a human-centric purpose. Some of those feedback loops will involve people's perception of the experience. What is good and “right” for some people can be bad and “wrong” for others. There is rarely, if ever, full consensus on what is right, good or best to do in a system.

Writing code can be complex. But it is often the conceptual work, people thinking together, that adds the most complexity.

Small changes in the way we think and communicate scale to big changes.

That's how systems work.

Bottom up communication structures, by themselves, aren't a magic bullet. They potentially create "many good but independent and uncoordinated ideas" (Brooks). Systemic change nevertheless depends on them and on changing some of our other entrenched cultural patterns.

In the book [Bad Blood](#), the true story of a billion-dollar fraud perpetrated by a Silicon Valley startup, we see some familiar technology culture patterns at scale:

- A disconnect between engineering reality and leadership demands. Communication runs down the hierarchy but not up. Essential feedback loops are missing.
- So many communication silos.
- A disconnect between Powerpoint presentations and reality. Product-driven demands reflect no concern for the system as a whole.
- The inability to solve complex problems together because trust, psychological safety and effective feedback loops, all necessary for systemic reasoning, are missing.

- Performance is measured by aggressive monitoring of hours worked rather than actual value produced. Forceful pressure is applied to “inspire” people to work harder or deliver faster. (It always surprises me how many people believe that force is inspiring.)
- Having to defend new or unwelcome recommendations to near exhaustion is normal.

Most of us have not experienced anything as extreme as Theranos, the company described in Bad Blood. That organization exhibited all the harmful patterns, all the time. But many familiar daily-life norms in technology work enabled their approach to appear rational and productive to many people for a too-long time.

“psychological safety allows for moderate risk-taking, speaking your mind, creativity, and sticking your neck out without fear of having it cut off—just the types of behavior that lead to market breakthroughs.”

—Laura Delizonna, Stanford University

In my experience, there is a direct correlation between successful teams that deliver difficult change on time and enjoyable relational experiences. The ability to think well

together is not only energizing and productive, it's a business critical skill. Exhausting teams overfocused on power and control have been my worst career experiences. Not because they sucked to work on (they did) but because they delivered very little meaningful change.

NOTE

You can't improve the technology system without improving the people system. And vice versa. From a systems perspective, they are one and the same.

Time is Always a Factor

“People assume that time is a strict progression of cause to effect, but actually from a non-linear, non-subjective viewpoint - it’s more like a big ball of wibbly wobbly... time-y wimey... stuff.”

—Doctor Who

We like to imagine time as linear and synchronous. We create Gantt charts and set milestones and take “next steps.” We manage time with calendars that structure our social attention. We track the time it takes to complete tasks. We are very, very

careful about our software's uptime, query and response times, and on-time delivery.

These are all helpful tools, when used with awareness. This book was written in a year-long series of structured “sessions”. I put on my noise-canceling headphones, start my background-music timer and write for forty-four minutes while ignoring (usually, mostly, kinda) mental urges to check email-slack-mastodon or do the Wordle. But the book also wrote me. Insights, thoughts and mental images arose at all hours of the day and night. Some of my work habits stopped working for me as I discovered that I do not always practice what I preach. Feedback from my colleagues or from Shira, my editor, would reveal a blind spot, a big blank spot that needed my attention. Off I'd go to explore it, totally “off schedule.” Writing is chaotic and if I hold the chaos at bay, I can't create something novel. If I surrender to the chaos, I can't create anything at all. Wibbly wobbly, timey wimey.

Linear, synchronous progress exists inside a nonlinear, asynchronous reality that is co-creating something new. That's okay, we don't need to fix that.

When I'm building software, I enjoy writing some code, running it, and seeing (immediately) what happens. Direct,

logical feedback. If it's broke, I fix it. Over the course of my career, this feedback loop has become increasingly time delayed. One event in one part of the system, eventually, triggered unwanted behavior elsewhere. Our IDEs can't catch that. Layers of caching, explicitly-designed time delays, made "seeing" what was happening in real time across a system even more challenging.

Similarly, the people process has changed from building a feature into a single codebase. Even when three engineering teams were pushing code to production, we did code reviews as if we were one team. We followed the best practices designed for that software (the way we wrote tests, for example) and the syntax rules were shared by everyone.

Now, different parts of the organization interact with different software or the same software but with entirely different goals, silo'd budgets and very little time or tolerance for designing systemic relationships.

Systems design is relationship design and relationship patterns are all about timing. Fifteen microservices or four teams working on different software parts, will be inherently asynchronous. Changes are happening asynchronously – the changes we intended and the changes we did not intend.

We often use the word “manage” in relation to time. We manage projects and people, complexity and infrastructure. But when projects and people act asynchronously, and in unexpected ways, we find ourselves in a muddle. Systems thinking shifts our attention towards “orchestrating”, a more subtle and artful approach. People and activities are viewed as interdependent and interrelated, like a symphony. The whole has cohesion and understandability even though the parts are played in their own time.

WARNING

Caution: I have heard teams describe asynchrony as the ability to be “independent”. For example, decoupling the front-end software from the back-end software so that the two teams can work independently. There is still a *relationship* between those teams and with the rest of the organization. Those relationships are part of a system.

We want to make boundaries around a software part, not build a fortress surrounded by a moat filled with crocodiles. Teams are still part of a community. The software they build is still part of a system. We still need to understand how the people, and the software, are interdependent. (If we are building fortresses and storming each other’s castles ... why?)

Independent and interdependent; manage and orchestration ... these are examples of the many conceptual shifts we make when we shift from software to systems thinking. In the next chapter, we'll explore more of them.

Summary

Systems thinking is relationship design. In systems, we are concerned with how relationships produce effect. We are concerned with the relationships within the tech and between the people. Systems are sociotechnical, there is no hard separation between people processes and technology. Both are constructed by our mental models.

We improve our mental models by creating conceptual integrity. Conceptual integrity is the most important consideration in systems design. Creating cohesion in our thinking process creates cohesion in our technology systems. This is challenging because of counterintuitiveness. In systems, we often recognize a trouble spot but make it worse with our solutions. Systems thinking practices help us navigate counterintuitiveness.

Navigate is a purposeful word choice – systems are always in flux. Time is always a factor and the impact of asynchronous but interrelated events are difficult to predict. In systems, we orchestrate, trying to create a harmony for the cacophony of many good but independent and uncoordinated ideas.

Practices

Systems thinking is strongly supported by two practices: free writing and modeling. Both are tools for understanding (and potentially transforming) thinking.

We will explore writing practices more deeply in Chapter 4. Here are some warm-up prompts. Set a timer for 10 or 15 or 20 minutes (your choice) and write your answers. Don't worry about grammar or full sentences or staying on topic, just follow your thoughts with your fingers.

- I have seen a lack of conceptual integrity, “many good but independent and uncoordinated ideas” in software and/or people processes. Some examples include ...
- Like Jay Forrester, I have seen situations where we knew where the problem was but our actions were making it worse. For example ...

- Right now, the most valuable sociotechnical process change we could make is I think this is the most valuable because ... (what are the reasons that convinced you?)

Modeling, in a linear world, has strict templates like UML and frameworks like TOGAF. I like and appreciate those tools, in the right context. But we don't need to follow exact modeling instructions to explore systems thinking. Over the course of this book, you'll add more model skills to your toolset. But let's begin with some free modeling (that's something I encourage you to always do).

Using a tool Miro or sticky notes on a whiteboard or a pen (digital or ink-filled), talk yourself through this prompt. Don't worry too much about how you "draw" things, just a little warm up practice using a basic system model.

- Remembering a time when you (and your teammates) had a goal, a change that you wanted to make to a system, use the basic system model in this chapter.
 - What were the ideas flowing in? (Draw some, make a list or create stickies to describe them.)
 - What was the output?
 - What was the process of resolving the discrepancy, moving from current state to goal state?

- Do you see any opportunities for improving the process you just described?

Questions for Your Journal

- Give an example of a situation where it was challenging to integrate “many good but independent and uncoordinated ideas”. Did counterintuiveness play a role?
- Describe ways that a software system, like a neighborhood, is more than simply the sum of its parts. What other factors are involved?
- Describe three examples of people processes that are designed to manage, solve or control software challenges. Are they effective? How might you change the people process to reflect a positive change in the software?
- Imagine you are shopping online for a new computer, including researching options. List eight ways that “time” is a factor in the software that supports this activity from start to finish.

Chapter 3. Shifting Your Perspective

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 3rd chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at sevans@oreilly.com.

Many times, I’ve heard people complain that systems thinking is “too abstract”. They want “concrete”; they distrust “ivory tower”. They resent thinking that isn’t delivered with code.

This reaction is a misunderstanding of systems thinking. Systems thinking is going deeper, underneath the surface-level

insights. It encourages us to shift our perspective so we can see what is actually happening and discover root causes.

In This Chapter

You will learn that systems thinking is going deeper, looking for the mental models that give rise to the events we experience. Systems leadership is becoming comfortable thinking with others, through modeling (whiteboards FTW) and core practices like respectful listening. The concepts we will explore include:

- Systems thinking can be structured by the iceberg model, going deeper into the patterns and structures that support our mental models ... and how they create the events we experience “on the surface”.
- Modeling, formally and improvisationally, is a core systems thinking practice.
- Prerequisites for developing systems thinking skills are constant learning, structuring inquiry, doing deep work (taking time to think), respectfully engaging with others and saying no to cat herding (aka enabling) whenever possible.
- Systems thinking introduces a new-to-us vocabulary that is still emerging.

- Many organizations face similar systems challenges (a fictional one is described).

The practices you'll discover are:

- Using the iceberg model, describe experiences you've had.
- Devise a few modeling exercises you might try – and try one.

The Iceberg That Sinks Our Initiatives

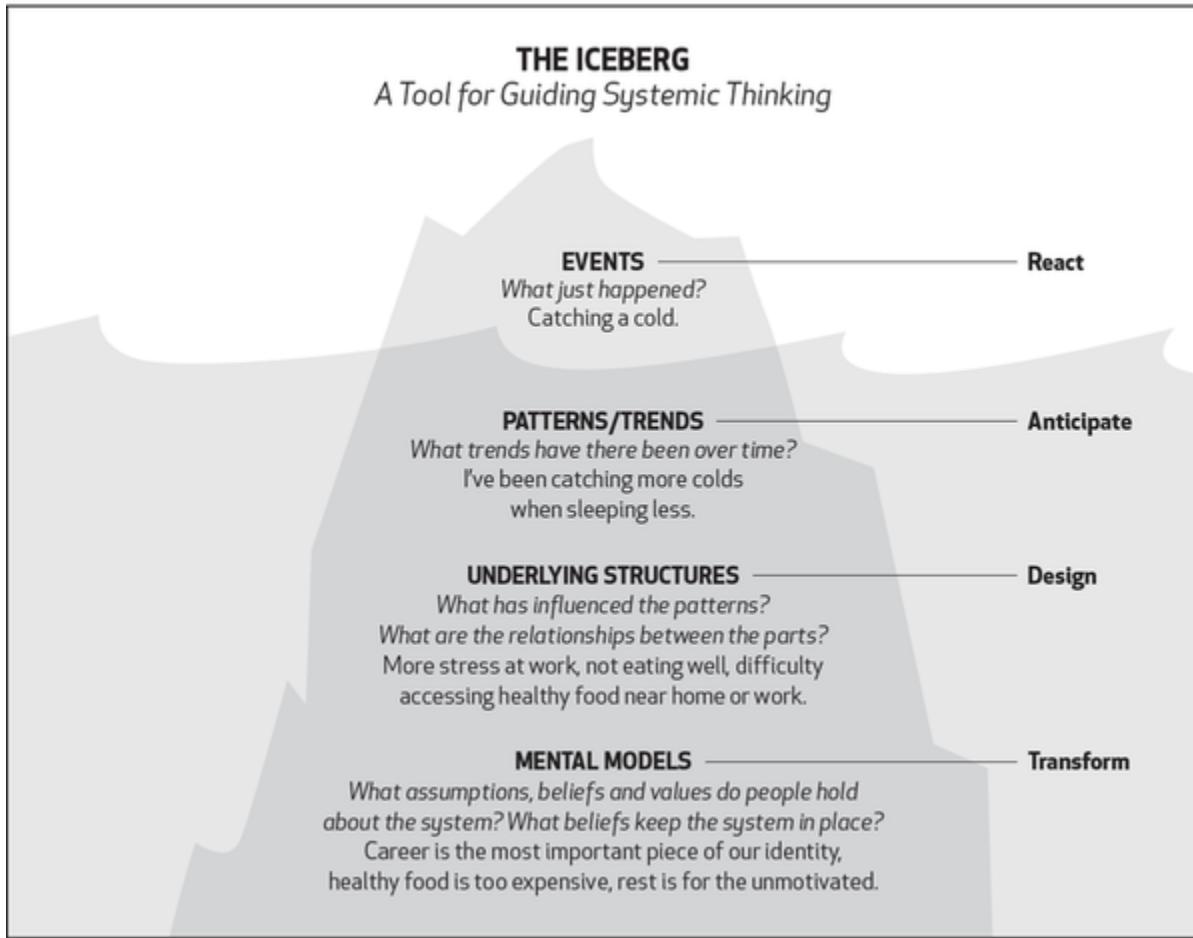


Figure 3-1. Figure caption to come

When we build software, we usually focus on the tip of the iceberg. What just happened? Bugs are happening, a response is slow, a feature fails to satisfy users. We fix, we patch, we plan better features. When those events reoccur, we take wider action to prevent them, with test coverage, for example.

Systems thinking goes into the underlying structures and mental models that give rise to events. What an event happens, we also think about:

- What has happened over time? (As I mentioned in Chapter 2, time is always a factor.)
- What are the patterns and relationships that support and structure these events?
 - The patterns can be in the software, in the relationship between the software parts, in the thinking and communication between people, in the world around us (nefarious user behavior, for example) and in our own minds. Often it is some combination of these things.
- What did we think, when we built this, and how did that thinking lead to this outcome?
- What do we believe, take for granted or presume about this situation?
 - How might those potentially unconsidered or, under the circumstances, outworn beliefs play a role in generating the event?

Here's an example:

Event: *What is happening?*

Faced with a novel systems challenge, the technology teams are failing to cooperate towards building better relationships between the software parts and the people thinking about them. (There's a lot of noise.)

Patterns & Trends: *What has been happening over time?*

When hiring, the organization has prioritized a technology-centric skillset that does not include “how well do you think with others to solve systems challenges?”.

Underlying Structures: *What has influenced these patterns?*

Whiteboard tests (in some form) are an industry-wide practice, organizations have adopted them as standard. The current team is made up of people who passed them and like them; they see no reason to change. There is no organizational support for learning, so they have to, for example, hire GoLang developers rather than hire experienced developers who can learn GoLang when they need it.

Mental Models: *What are the relevant assumptions, beliefs or values?*

Experience with a specific technology tool predicts the quality of knowledge work that will be delivered by the person in the future. People best qualified to develop technology solutions can improvisationally solve coding puzzles, on demand, during interviews. “Soft skills” like communication is not a priority for software engineers (that’s what project managers are for.)

Some of the best engineers I've worked with have failed whiteboard tests. Three times, I saw orgs almost-pass-over a candidate for being “not technical enough”, decide to hire them and six months later, that person was the most valuable engineer on the team.

Conversely, some of the most difficult engineers I've worked with were top experts in the technology stack we used. (They also blocked the adoption of tools or languages they weren't expert in using.) They were the ones who made up the tests others had to pass. Principal, as a career aspiration, has not necessarily meant someone with good systems thinking skills, which is a blocker in a world full of systemic challenges.

Like everything in systems, the mental models that justify and support hiring practices are not, by themselves, a problem (necessarily). Problems arise when the relationship between events, patterns and structures arising from our mental models, in particular circumstances, push us in the wrong direction. Then we need to go deeper and rebuild our thinking from the bottom up.

In our example, the organization did not prioritize communication, cooperation and systems thinking skills. When

it needed those skills to succeed, it floundered. This is a solvable problem. But two things usually happen instead:

- The organization makes the problem worse by hiring more project managers to control the situation. The problem was too much noise because people can't think well together, adding more noise will not improve the situation. More project managers may in fact be part of the solution but by shifting the focus onto a fix, rather than understanding the root cause, the root cause remains and will continue to structure the entire situation.
- The people in the organization blame (a lot). And they blame the wrong thing. Lack of leadership, tech blames product, product blames tech, not enough money, too much focus on the wrong thing, lack of Kubernetes, too much complexity, etc. As we'll see in chapter 5, there is a lot of blame in systems. We rarely understand how the structure of our thinking has led us exactly where we ended up.

The iceberg model shows us that systems thinking is about shifting perspective. Thinking in systems is changing your perceptions – the ways in which you see, understand and interpret what is happening. The practices and tools you need to accomplish this are the ones that support your ability to change your mind.

Modeling as a Core Practice

The most valuable approach to understanding what is happening and exploring a shifting perspective is modeling. In systems, you do a lot of modeling. Modeling is one of those words that means different things to linear and nonlinear thinking. In linear thinking, models are concrete pictures, like instructions that come with Ikea furniture. Those models are helpful!

In systems, planning and documentation are not what we mean by modeling. Modeling means using tools to think together. My favorite are whiteboards (or digital equivalents like Miro).

Many other tools, especially analytics tools, can be used to help us think in systems. A colleague and I have begun developing workshops on tools for systems thinking, because there is so little support for software professionals using tools in a more creative, nonlinear way.

Modeling is dismissed by too-many software teams as a waste of time. They prefer to “go fast and break things”. “We don’t do big upfront design.” “The code is my model.” This blocks collective reasoning, which is critical in systems. But more importantly, it’s a misunderstanding of why modeling is valuable.

Modeling as a method of inquiry, a way to learn, think, understand and see things from multiple perspectives. We can model with words or pictures or code or all three. We can model alone and we can model together. Models make our thinking and understanding visible and explicit. We use models to test our conceptual integrity. Do we understand what is happening and the impact of change?

All models are defeasible, they become obsolete when we know more or the circumstances change. Models are thinking tools, not reality. Five people solving a problem will arrive at a more-reliable solution faster if they “think out loud” through modeling. The reason we struggle to find common ground is because our mental models are different. We are often using the same words to mean different things. Modeling makes thinking mutually visible. And reflects the “stuck spots” where we don’t see things the same way. When we find them, we can dive deeper to understand why.

Here is an example scenario: The system’s purpose is to provide essential information to a user when requested. The delay between request and return is too long. What are some modeling activities that can help us?

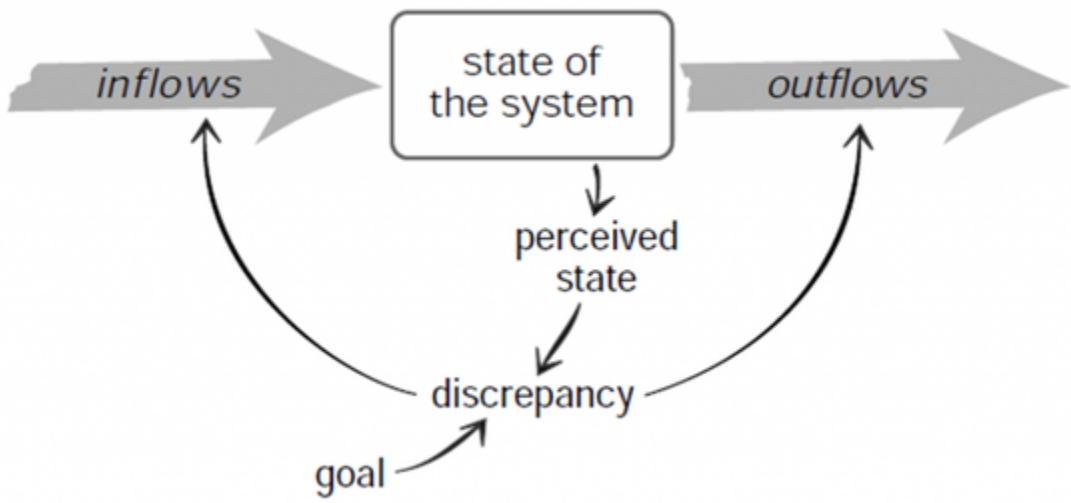


Figure 3-2. Figure caption to come

System concept	Questions we might ask	Models we might use or make
What is the current state?	What is happening now? How does information flow in and how does it flow out? What is happening when information is at rest and in motion?	Event storming, data architecture, domain models, information flow diagrams

System concept	Questions we might ask	Models we might use or make
What is the goal?	<p>How fast should the information be received? Why is “faster” valuable – will it enable the system to serve its purpose? How do we know that changing this discrepancy is “better”? Better for who and in what ways?</p>	<p>Single page description of Why, What, Who, How that connects the work to the purpose. Data about the system and user experience. Prioritization and business process models.</p>

System concept	Questions we might ask	Models we might use or make
What is the discrepancy?	<p>What is the time difference between current time and goal time? Under what circumstances does it occur? How do we measure it? Do our measurements actually measure the problem we are solving?</p> <p>What is impacted by the delay? What happens during the delay and how it is being reinforced?</p>	<p>Working code, reporting, data, analysis of data,</p> <p>downstream impact</p> <p>modeling, user-reported information,</p> <p>feedback loops / time delays</p>

System concept	Questions we might ask	Models we might use or make
What is the perceived state?	<p>We say “too long” but what benefits from the delay?</p> <p>What are the tradeoffs involved in making it faster? Does everyone perceive this as a “problem”? If not, why not?</p> <p>If making it faster has impact elsewhere, who will perceive those changes as harmful or helpful?</p>	<p>Event modeling, decision-making history, impact models (if we do this, that will happen), balancing and reinforcing feedback loops (current and future), stock and flow models</p>

NOTE

When we think in systems, we are looking for answers by figuring out, “what are the right questions to ask”? Our questions are rarely only about the code.

In later chapters, I'll suggest more modeling activities to try. But it's important to remember that you can just dive in and make models. Yes, recipes help, but you are a knowledge chef. You can create your own recipes and modify them as you learn.

Five Core Practices

In addition to modeling, there are five core practices that I hope we agree are necessary in our line of work. At least, I hope they don't set your hair on fire. Technology folks are a learning-thirsty bunch, having chosen a work in a constantly changing field. I feel safe in assuming these are, to some extent, already a part of your life.

1. You are constantly learning.

You don't mind discerning what to learn and how you'll learn it.

2. You're comfortable structuring inquiry.

You don't mind figuring out how to figure something out. Being in a room with three people, considering a problem none of you have faced before, is a good day.

3. You do, or are willing to do, deep work.

By deep work, I mean distraction-free time invested in becoming aware of, structuring, strengthening and improving your thinking. Concentration is key. However it happens — daily, weekly, regular retreats — focused, generative work happens. (If you've moved into a role that fills your days with meetings, I encourage you to begin now making some space for contemplation.)

4. When others share their thinking, you engage respectfully and help to strengthen it.

When you disagree with someone, instead of negating, belittling, or dismissing, you help strengthen their thinking with relevant, insightful contributions. You don't cling to your opinions, you are open to learning through thoughtful reasoning.

5. You are willing to stop being Sisyphus.

In Greek mythology, Sisyphus was forced by the god Zeus to roll a giant boulder up a hill. Every time the boulder neared the top, it rolled back down. Sisyphus rolled that rock for all eternity. People with nonlinear skills are sometimes playing “glue roles”, holding “parts” together ... people, teams and software parts ... that don't hold themselves together. Cat herders, we call them. In

nonlinear approaches, nobody gets to be a cat. Cat-like refusal to think well with others needs less “gluing” and more behavior modification. Our goal is to change patterns, not enable them. (With the caveat that occasionally, of course, gluing and cat herding is required. The emphasis is on “occasionally”.)

Perhaps you agree with these core practices and are excited to read more. Great! Keep in mind: Agreement, sharing the same opinion, is not our goal. Our goal is to practice – to weave new behaviors into the fabric of our daily work life. How are you practicing these core skills already?

Learning a Different Language

As technology professionals, we share a vocabulary that we use when we talk about software. We use common words to describe principles and qualities that we know, from experience, are good. At conferences, we put those words on slides. We understand, for example, that “loose coupling” is usually preferable to “tight coupling”.

Systems thinking introduces an expanded, and sometimes contradictory, vocabulary. The words can be confusing,

especially if you aren't already comfortable with conceptual work. In systems, we sometimes use the same word but mean something different.

For example, we know that “learning from our results” is important. We “iterate and improve” based on cause and effect. The word learning has a different quality in systems thinking. Yes, we gather data to test the outcomes. But when we say “learning”, we are describing the nature of the work itself. Learning-driven teams are systems thinking teams. We also mean “everything is knowledge” and as knowledge workers, this is our craft. We also mean “model the feedback loops”.

This might sound vague and confusing. Don't worry, we will dive deeper into learning in chapter six.

Here are some examples of vocabulary partners that reflect the somewhat-different concerns of software and systems.

Software	Systems
Incremental	Emergent
Linear	Nonlinear
Rule out the bad ideas	Generate insight
Code review	Collective reasoning
Empirical	Coherence
Iterative	Feedback loops
Improvements	Leverage points
Best practices	Conceptual integrity
Discovery	Counterintuitiveness
Feedback driven	Learning driven
Experimental	Observational
Control	Self organizing
Boundaries	Interdependence
Focus on parts	Focus on relationships

Software	Systems
Linear processes	Patterns
Users	Context
OKRs	Purpose
Technical	Sociotechnical
Documentation	Frameworks
Features	Capabilities
Reduce & manage complexity	Understand & model complexity
Stateful	Dynamic
Concrete	Abstract
Reductionistic - break into parts	Holistic - sum is greater than the parts
Separation of concerns	Intersection of concerns
Modular	Interconnected
Inspect & adapt	Delays over time

Software	Systems
Platform	Stocks and flow (of information)
The parts	The whole
Delivery	Orchestration

Many of my colleagues and I agree that the struggle to define concepts is, currently, the heart of systems thinking in the technology world. We confuse each other with words even when we mean the same thing. We haven't established a familiar vocabulary to describe nonlinear approaches.

The word Agile, for example, triggers different meanings for different people. Our definition of Agile is inextricably linked to our experiences. Agile, to me, means a light-weight process supporting strong teams that deliver high-value changes, as they see fit to build them, with little need for managerial control. For others, Agile means endless JIRA tickets, silo'd workflows or SaFE. When we are designing a delivery experience that supports us, we need to first ensure we are imagining the same experience.

Here are some examples of common challenges we experience in systems and some vocabulary that might help us discuss

them:

Creating self-sufficient and self-organizing teams who can tackle complex challenges without need for too-much hierarchical control

- Systemic reasoning
- Sociotechnical systems
- Self-organization
- Learning teams

Creating a shared understanding of a situation and developing approaches to change

- Mental models
- Collective reasoning
- Collaborative modeling
- Heuristics

Improving individuals ability to deliver matterful, challenging work and grow essential skills

- Deep work
- Personal mastery
- Integrative leadership
- Systems thinking

Why organizations don't see systemic problems and/or are making them worse

- Counterintuitiveness
- Leverage points

- Command & control approaches
- Reinforcing feedback loops
- Lack of focus on patterns

Inability to make tradeoffs and discern where to invest time, energy and attention

- Nonlinearity
- Top-down (rather than bottom-up) hierarchy
- Start with what instead of why
- Feedback loop design
- Lack of conceptual integrity
- Not designing for emergence

Inside of our linear mindset, we believe we understand words like system or feedback loop or conceptual integrity. When we have a nonlinear mindset, those words take on critical nuances

that are essential to understanding what they really mean. When I use a key phrase, I will define what I mean. But I encourage you not to overthink the vocabulary. The practices and concepts will help you regardless of whether or not we agree on the name. Remember:

There are only two hard things in Computer Science: cache invalidation and naming things.

You will need a few more vocabulary words as you read:

We will practice *systemic reasoning* by creating a holistic understanding and well-reasoned support for an action, idea or theory. Systemic reasoning trains our minds to think well, with others, even when we can't know if we are right. We synthesize knowledge, experience and sound judgment into ideas based on valid reasons. Our *reasons* are the considerations that justify or explain our thinking.

Feedback loops are the ways that outputs in a system circle back as inputs. When we push code designed to increase user interaction by 25% and user interaction increases by 25%, that is a feedback loop. When user interaction increases but in a hacky way we didn't intend, that's a feedback loop. When the

code takes down the entire system, that too is a feedback loop. Processes like autoscaling rely on feedback loops.

Emergence is the process through which simple interactions among individual parts form complex behaviors and patterns at the systems level. We design for emergence when we design rules for these interactions. Performance, for example, is a systemic property, describing how well systemic relationships scale.

“to deeply assess contexts, to truly read undercurrents as well as surface activity, to not miss emerging correspondences between seemingly disparate things, we need to talk about empathy as a skill.”

—Ann M. Pendleton-Jullian, *Design Unbound*

Systems thinking includes *empathy* as a critical skill. Empathy is not simply emotional caring – it is understanding that you are cognitively and emotionally embedded in the systems you are designing. When you engage in modeling with others or systemic reasoning, you practice cognitive empathy, which is the ability to imagine the situation from someone else's perspective.

Uncertainty (being unable to wholly predict the future) and its close cousin *ambiguity* (inexactness) are not curse words in systems thinking. We can't be certain of the future. We can't be concrete in every communication. For example, systems thinking will almost always rely on metaphor, to some extent. In systems thinking, we construct our best possible solution under the circumstances despite uncertainty. We continuously learn from the outcomes.

Emergence and uncertainty are daily life in *complex* systems. As complexity increases, parts and relationships interact nonlinearly. You can no longer understand the system by deconstructing its parts, you also need to understand patterns. The internet is a complex system.

When does a technology system become complex? For our purposes, increasing complexity arises when:

- The impact of small changes is difficult to predict. For example, the social, political, attentional, and psychological impact of adding a “like” button to a social media platform.
- It is difficult to model the system in any meaningful, static way. Models need to be in relationships to each other to describe the system.

- Behaviors are emerging that were not explicitly designed for, as a result of relationships in the system. (Sometimes we call these “bugs”.)

I encourage you to be careful with your vocabulary and take the time to define key words. In software, we have our own language and understand each other. We’ve accepted as normal a lack of understanding between, for example, tech and marketing or business and SREs. In systems, we need a shared vocabulary in order to cultivate conceptual integrity. Taking the time to agree on naming is worthwhile.

Mago’s Quandary

Throughout this book, I will describe a fictitious organization with increasingly-common, real-world systems challenges. Meet Mago.

For decades, Mago published the most popular, internationally-distributed magazine in the world. Their state-of-the-art publishing software and distribution system ensured each edition reached millions of people worldwide, on time, every Sunday.

In 2010, Mago launched a website. Over the next few years, everything in print was also shared digitally, their articles,

graphics, ads and collections highlighting timely topics. Like most organizations, they built their digital presence with a single piece of software then extended, customized, and scaled it by adding lots of caching. They encouraged the world to come view the information available at a URL. Page views quickly rose to millions per day.

Mago, like most organizations at the time, translated a printed page into a web page. Their data architecture, delivery workflows, and software choreography revolved around the concept of a “page”. HTML was invented to structure digital pages. Mago didn’t realize that the emerging global information system that is the internet would shift the paradigm. Today, what does “page” mean in a world where people share content everywhere?

An entire ecosystem of organizational infrastructure arose during this shift. Subscription transactions, tracking and analyzing reader behaviors, monitoring system availability, morphing assets (like images) into varying shapes, sizes and types. Business processes digitized, payroll, hiring, planning and hosting events (something Mago was known for). Communication tools became inextricable from productivity, enabling co-located teams to meet, plan work and track progress.

The world around Mago's website became increasingly interconnected. They needed to also show up on search engine results, social media platforms, news aggregators, video and audio platforms. The world, not just the website, needed a steady diet of content. Discussions about published articles, which Mago hoped to track, moved away from comments on a page and happened wherever people chatted.

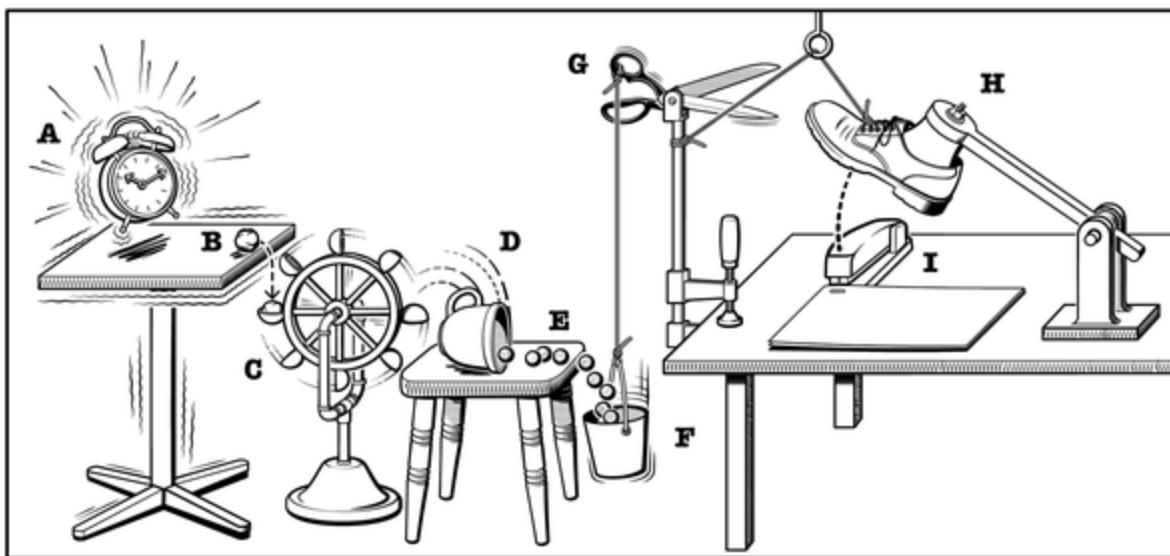
People accessing information on their desktop wanted different information than people on their phone. Mago built an app, then another. Soon that wasn't sufficient. People expected information to morph in the browser depending on the device they used to access it or where they were geographically. The demand for information *in context* was quickly becoming the norm.

As bandwidth increased, so did the demand for multimedia content. Mago expanded staff to create extended video stories, podcasts, and custom interactive graphics to show data trends. Each of these innovations required software systems to support them.

The biggest change was the increase in asynchronous workflows. Publishing no longer fit into a weekly, or even a daily, rhythm. Various content and the software supporting it

went through various delivery workflows. More people needed to keep these workflows in sync as the foundational systems structure, delivering pages, was quickly eroding. The shape of information was transmuting and increasingly ubiquitous. The shape of everything was changing.

To keep up with the relentless pace of modern technology, the Mago team built a lot of software. As it became increasingly difficult to continue extending the original, now legacy, digital software, product people went rogue and built separate websites for new content. The software created, over time, an ad hoc collection of mostly dissociated parts. Very few people could name all the parts or knew how they functioned. The system looked like a Rube Goldberg machine, with production people moving data by hand.



© Vernier Software & Technology

Figure 3-3. Figure caption to come

As relational complexity increased, it became clear that continuing to expand the current system was not going to work. How many pieces of software can be duct taped together? The paradigm had shifted. Mago's Quandary became: How do we design a system that meets the demands of the modern systems age?

As we explore systems thinking and nonlinear approaches, we will come back to Mago's Quandary.

NOTE

If you happen to know that I've architected systems for organizations like *The Economist* and The Wikimedia Foundation, you might wonder if Mago is a thinly-veiled tell all. While the fictitious examples you'll read in this book are inspired by real-world experiences, they reflect modern patterns and challenges faced by many, if not most, information systems. Any resemblance to actual people or specific organizations is simply because everyone is trying to figure out how to evolve into the systems age.

Mago's Quandary: Examples of Nonlinear Approaches

Explaining the difference between a linear and nonlinear approaches is tricky – because it depends on your mindset. Teams can be doing the exact same behaviors with very different intentions and thus, very different outcomes. But workshop attendees have found it helpful to draw a map, however sketchy and incomplete, of some examples.

Here's the challenge: if you are familiar with information systems like Mago (or perhaps even if you aren't), you might want to argue with these. You might interpret this list as "the wrong way and the right way." I am *not* saying the nonlinear approaches are the RIGHT approaches. It depends! I am saying

that, In Mago's case, here are some nonlinear approaches that might help them:

Linear	Nonlinear
Model the solution that meets the need of a new channel or consumer planning.	Understand systemic pain points. Model the underlying capabilities that will support delivering content to future customers.
Decouple by designing an API that shares the “page” with other software parts.	Create a canonical data model that structures semantic naming of the information distributed by the system. (Understand the system’s information sharing).
Add an API that shares all information on a “page”, the consumer can parse it to use what they want.	Structure information that is inherently interrelated and queryable, then allow consumers can ask for what they need.
Build services that leverage information from existing systems (through APIs or shared databases).	Design an event-based system that supports adding new services without direct integrations (loose coupling).

Linear

Nonlinear

Replace old software parts.

Reconsider the context in which those parts operate now – can we form relationships between parts rather than rebuild a monolith?

Lift and shift migration to the cloud.

Cloud native: Rebuilding to maximize the benefits and opportunities of running in the cloud.

Design one-off features to meet short-term needs.

Design capabilities that will support the current feature needs as well as potential future needs.

Decouple front end and back end.

Build an information service that can gather information from backends and serve front ends that haven't been invented yet.

Linear	Nonlinear
Put an abstraction layer in front of legacy software to “modernize” or build a new green field system to avoid making changes to a complex legacy system.	Use an abstraction layer to iterative redesign software parts and bring them online, creating eventual obsolescence (strangler pattern).
Focus two teams on different initiatives.	Focus two teams on different parts that reflect the domain.
Get everyone’s opinion on decisions (or make all the decisions without getting feedback).	Collective modeling, systemic reasoning and separation of concerns (so “who owns the decision” is clear from a systems perspective).
Glueing together applications by adapting the outputs of established tools owned by other parts of the organization.	Reviewing the tools used across the organizational boundaries to ensure the tools integrate well.

Linear

There is a problem with a system not scaling, we identify an issue with the number of connections the system can handle. We increase the number of connections and the failure point shifts elsewhere.

Nonlinear

Faced with a scaling change, we explore the problem holistically – including the human behaviors leading to the issue. We identify multiple areas where the system would fail to scale but ultimately address a behavioral issue leading to sudden spikes of activity.

Mindshifts are Hard

Before we dive into the individual, group and leadership practices that will support your systems thinking, a pause

This is difficult work.

Systems thinking is most valuable when the paradigm shifts, when the purpose of a system is changing. We are facing that in my software situations. Before you can design those systems, you have to shift our own internal paradigm. You need to see potential changes for yourself. And you'll need to think well

with others, some of whom aren't the slightest bit interested in systems thinking.

We aren't simply tweaking our thinking with these practices. Our thoughts and experiences and conceptual models and feelings and communication patterns and the world around us are all entangled. Detangling is deep work. I want you to know that you are not alone. This work matters to do. We are faced with so many system challenges nowadays, we need you. We need you to consider alternative approaches and perspectives, even when it would be easier to think and act linearly.

The upcoming practices might sound easy. I hope you find them straightforward enough to try! But nothing about thinking differently is easy. It's hard to change, to be the voice of reason while adrift in a sea of noise, trying to make some sense of things. I trust that we are up to the challenge. I'm with you, as you move through this, if it's only as someone you know who has battle scars and a deep desire to sally forth regardless.

Summary

Systems thinking has a bad rep as “too abstract” when in fact, it is going deeper, under the surface of the events we experience.

You can practice going deeper by using the iceberg model, considering patterns and structures that arise from our mental models ... and how patterns, structures and core beliefs generate the events we experience “on the surface”.

Modeling is a core systems thinking practice. Adopt the habit of modeling thinking, alone and with others, whenever a challenge arises. You’ll discover that you illuminate more of the relational impact when you do.

This book presumes a few prerequisites: constant learning, structuring inquiry, doing deep work (taking time to think), respectfully engaging with others and saying no to cat herding (aka enabling) whenever possible. If any of those practices are new to you, now is the time to try them!

The vocabulary to talk in systems is still emerging. Often, we use the same word in linear and nonlinear approaches but we mean very different things. Defining new concepts when you use them is a great way to create shared understanding.

Many organizations are facing similar information systems challenges. This chapter introduced a fictional example, but you can use whatever circumstances are familiar to you. Consider

the ways that nonlinear thinking might help create healthier, higher-impact approaches.

And, btw, systems thinking is hard.

Practices

Free writing prompts:

- When I've tried to think in systems, alone or with others, the biggest challenge I faced was ...
- Considering the five core practices, I am best at ... and need to improve my skills in ...
- The modeling experience that had the most benefit was ...

Modeling:

Using the iceberg model, consider three pain points in your current situation. Dive deeper and model the patterns and trends, underlying structures and mental models that lie beneath the obvious events.

Choose one of your examples. Set up a digital whiteboard that you can share with others showing the iceberg model. Gather a few people (virtually) together who experience the event from different perspectives. Model the challenge using the iceberg

structure. How was the group outcome the same as, and different from, the one you created alone? (You can totally do this in person on an *actual* whiteboard.)

Questions for Your Journal

- What are the four levels of the Iceberg model? Give three examples of recurring events that people have unsuccessfully tried to solve. Describe them using all four levels.
- What are the five prerequisite practices described in this chapter? Are there others you would add? Describe how these skills are part of your daily life now.
- Give five examples of “systems thinking” vocabulary and their definitions.
- Describe three outcomes you’ve experienced that were framed in a linear way. Would they benefit from a nonlinear approach? What would you recommend?

Part II. You Are a System of Thinking

Chapter 4. Self-Awareness as a Foundational Skill

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 4th chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at sevans@oreilly.com.

Self-awareness is a trait—or maybe “practice” is the more accurate way to put it—that everyone can always improve at. It is part emotional intelligence, part perceptiveness, part critical thinking. It means knowing your weaknesses, of course, but it also means knowing your strengths and what motivates you.

—Neil Blumenthal, Co-Founder of Warby Parker

You cannot improve your thinking if you aren’t aware of your thinking.

Systems thinking is grounded in metacognition: critical awareness of your own thought processes. When you reach a conclusion, do you know how you got there? Metacognition isn’t simply “thoughts,” it is also noticing your reactive experiences and understanding how you learn (change your mind) best. In the next two chapters, we will dive deeper into reactions and learning. In this chapter we will explore self awareness, become aware of our own thinking and the patterns that direct our thoughts.

For some software professionals, this part of the book will challenge their definition of “technology skills.” We have been

taught to focus on *what* we know, rather than *how we think, feel and learn*. In systems thinking, our minds are our instrument, our ability to observe is critical to the work. Our goal is to improve the trustworthiness of our insights. We can not do that work well without self awareness.

When we are self aware, we can objectively interpret thoughts, feelings, experiences, choices and actions. Strong objective interpretation is the foundation on which we construct the thinking that we share with others.

Self awareness incorporates more than our thoughts, it interrelates cognitive patterns, feelings, physical sensations, behaviors, future expectations and past experiences. We are an embodied system, not a dissociated mind, our thinking happens as a result of systemic patterns that we experience. The best way to begin practicing systems thinking is to practice with your most intimate system: you.

Many, if not most, of the patterns you'll find in yourself will mirror the patterns you experience in the world around you. For example, I find it painful when someone blames me for something that is not my fault. Yet, my own mind jumps to blaming others before I've taken time to consider the veracity of that reaction. Sometimes, your thinking is strong. More often,

it is reactive, fallacious, habituated and lacking conceptual integrity. That's okay; that's why we practice.

Without self awareness, you can't create conceptual integrity. If you pay careful attention to your thinking process, you'll see that you are being mentally dragged around by your unconsidered thoughts and reactions. Without self awareness, you also can't overcome counterintuitiveness, the habituated solutions that feel "right" to us rather than discovering the unfamiliar solution that actually solves the problem.

Counterintuitiveness, if you remember from chapter 2, is a common systems thinking experience.

As you will recall from the iceberg model, systems thinking is, at the core, working with mental models. It is the system of generating concepts that gives rise to our technology systems. As we become aware of our own mental models, we become more skilled at seeing them in the world.

You will not maintain physical strength while lying on the couch eating ice cream and binging Netflix, The same goes for self awareness - it is a strength developed by exercising it. Which is why we need to continuously practice, paying attention to what we consume, both physically (like ice cream) and mentally (like 9 seasons of Suits).

As you'll see later in this chapter, you are not judging yourself when you practice self awareness. I've watched 8 seasons of Suits on Netflix. You are creating nourishing and supportive habits for your knowledge work, first by becoming familiar with your own thoughts

Remember our simple system model?

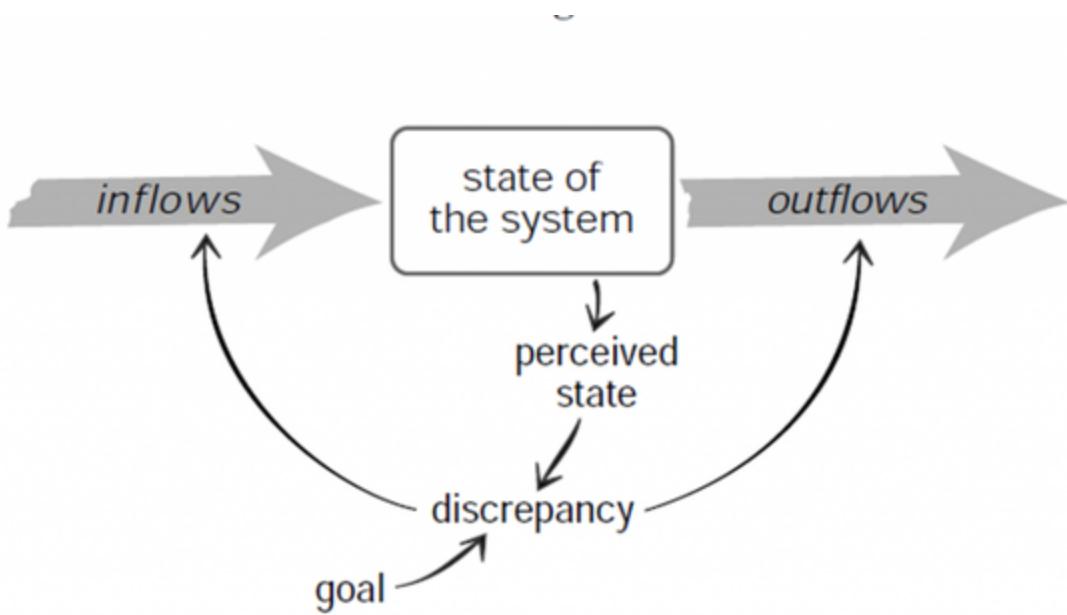


Figure 4-1. by Donella Meadows, <https://donellameadows.org/archives/leverage-points-places-to-intervene-in-a-system/>

Each of us is a physical system that consumes information from external and internal sources. Our mind interprets that information and responds based on a myriad of factors – environment, chemistry, genetics, upbringing, what we ate for breakfast, etc. Our thoughts generate our actions and vice

versa, in reinforcing feedback loops. Everytime we experience pleasure from eating ice cream, for example, we are more likely to associate ice cream with pleasure. Every time our ideas are ignored or dismissed, we are more likely to doubt them.

As a system, we are more complex, conditioned and unable to “control” our thinking than we imagine. We have goals and, when we practice self awareness, we discover that our true goal is operating under the radar of our conscious awareness. I often have thoughts like “I need to skip writing today and go to the grocery store.” My goal, on the face of it, is to get groceries because I’m on the edge of starvation. My real goal is to avoid the discomfort of writing and replace it with the distracting pleasure of ice cream. Without awareness of these processes, we are carried along in their chaos.

In Part Three, we’ll explore more-specific methods of constructing thinking including systemic reasoning. First, we get to know our own thinking system.

In This Chapter

You will learn to become aware of your thinking and experiences. Self awareness is foundational to nonlinear thinking. Systems leaders with mature self-awareness skills are

more balanced and stable when faced with the morass of conflicting ideas around them. The concepts we will explore include:

- You can't improve your thinking unless you are aware of your thinking. Metacognition is the practice of knowing your own thought process and patterns.
- Free writing, by hand, is an excellent tool for developing metacognition. (There are others.)
- Self awareness isn't simply seeing what you think, it is also uncovering your blindspots by proactively seeking out what you don't know you don't know.
- The quality of your recommendations and decisions depends on "being onto yourself" and knowing when your thinking is stuck in a rut.
- Self awareness itself is a great teacher.

The practices you'll discover are:

- Writing every morning, first thing, improves your self awareness (and, over time, your ability to think in systems.)
- Other practices like walking, meditation and discourse are also self awareness boosters.
- Improve your understanding of how you make decisions.

Notice Your Thinking

There are many fascinating books, courses and other resources describing the science and complexity of self awareness and metacognition. Don't start there. Instead, start by looking at *your own mind*, its patterns and processes. Getting swept up in interesting theories and ideas is easy. Paying attention to your own thoughts, feelings and experiences—that's hard.

Systems thinking expands our capacity to do hard things. Perhaps you imagined the hard things were complex, event-driven software systems or increasing layers of abstraction. Those are hard things! Self awareness helps us do them. The more comfortable you are with the ambiguity and uncertainty in your own mind, the more comfortable you will be immersing yourself into complex situations that have no right or controllable solution.

Nonlinear approaches always begin with observation—paying attention to how things work. When I begin learning about a new system, I begin by modeling the current software, describing the flow of information and the stuck places. I listen to the frustrations people express because those frustrations

point me to the stuck places, the leverage points. I ask, learn, see, understand.

(Important note, I would not need to do this, to the extent I have, if there was already an insightful description and model of the system and how it serves its purpose. I have yet to engage with a software system that has this systemic view available to peruse. As part of your learning systems thinking practice, you might want to create one.)

We continue to observe throughout the delivery process and the whole lifecycle of the system. Observation never ends. But as we do, we are still looking at what is here, now. What is actually happening? How are the patterns reinforcing (or not) our goals? Continuous observation will show you what you need to learn before you act.

We don't begin with trying to "fix" our thinking because if you trying to fix something without awareness will make a mess. Perhaps you've experienced this in your professional life? The new boss who wants to transform the software without knowing anything about it? The new silver bullet, Kubernetes or continuous deployment, that fixes some problems but creates more? Without awareness, your fixes are New Year's resolutions that are abandoned before Groundhog Day.

Who is the “you” that wants to control your thinking or fix yourself? That part of your mind is extremely steeped in linear processes. Those are not the processes we want to strengthen. We want to strengthen our ability to notice, understand, listen and see clearly. We want to spot patterns, blindspots and habitual processes.

We can listen to what the system tells us, and discover how its properties and our values can work together to bring forth something much better than could ever be produced by our will alone.

We can't control systems or figure them out. But we can dance with them!

—Donella Meadows, *Thinking in Systems: A Primer*

Without self awareness, when we dance with systems, our unexamined thinking and emotional reactions will color, block or reconstruct what we see and hear in the systems around us. We will cling to outworn ideas or jump on bandwagons that lead us nowhere. Our thinking impacts everything we build, every team we join, every meeting we are in, everyone who works with us. It is the stuff that knowledge work is made of.

The First Practice

There are many ways to cultivate self awareness but one practice has been, by far, the most valuable to my work. You might love it and, like me, do it every day for the rest of your life. You might discover it's not for you and try something else. You won't know, though, unless you *try* it. So I am encouraging you to not deliberate, not to imagine whether or not this is a matterful practice. Instead, try it, see what happens. Let the experience reveal (or not) its value.

For one week, wake up every morning, pick up a pen and paper, set a timer for 20 minutes and write.

Write whatever comes to mind. If you get stuck, write "I don't know what to write" until more thoughts come to you. If you think "this is a dumb exercise", list all the ways it's a dumb exercise. *The only rule is: keep your hand moving.*

Sounds simple, yes? Perhaps but the 17,659 reasons you come up with not to do this practice every morning might make it harder. If you can't do 20 minutes, do 10. If you can't do 10, do 5. Notice, and write about, all the things you should be doing instead. Why you hate it, why you resist. Or why you love and need it and why its worthwhile. Write letters, lists, notes to your

cat. It doesn't matter what you write about, only that you create the space to share your mind and observe it as you do.

Cal Newport, bestselling author and CompSci Professor at Georgetown, argues that daily solitude is essential for knowledge workers. He defines solitude as "isolated from input from other minds". Solitude is the primary benefit of morning writing. We train our attention on our own thinking. We'll talk more about solitude in Chapter 6.

Morning writing, like exercising, creates a physical rhythm designed to strengthen our body and increase flexibility. Morning writing strengthens our self awareness, improves our conceptual integrity and mobilizes our thinking to flow. Doing this practice every morning creates space in your life for *your* thinking. And connects a physical experience (writing by hand) with thoughts and their flow.

When the week is up, commit to 30 days.

You probably have a lot of questions. Do I have to do this in the morning? Do I have to write by hand? Yes, morning, yes, by hand ... keep it simple and let the practice reveal its benefits.

I think self-awareness is probably the most important thing towards being a champion.

—Billie Jean King, One of the Greatest Tennis Players of All Time

Here's an example of a situation where my writing practice had a major career impact.

At one organization, I was buried under a pile of chaos; leadership changes, circuitous and endless disagreements, maddening and unworkable “new” strategies, derisive words being said in meetings ... it was a mess.

There were also good strategies, helpful supportive colleagues and matterful work to be done. I could no longer tell the difference between what mattered to do and what was a waste of energy. Thoughts, opinions and emotional reactions (like feeling powerless) were tumbling around in my mind. I felt paralyzed and powerless.

During my morning writing practice, a question occurred to me. “What if I were in charge?” What if I were the incoming CTO, someone who could say or do whatever was strategically sensical? I wrote out exactly what I would do, including what I

would say to the person I would remove from their role as a result of their behavior.

I listened to myself

Through this exercise, I discovered a path to insight. I didn't show anyone what I wrote, I didn't need to—I knew what to do. Eventually, I moved on but not as a shameful reaction, as a solid choice to navigate in a different direction.

Alternative Practices

Morning writing practice is helpful for most people. Are you one of them? I've been teaching workshops, using writing as a thinking practice, and attendees have found this practice surprisingly beneficial. Try it (starting tomorrow) and see for yourself.

If you decide you want to try something else, or you'd like more than one practice, here are more recommendations.

Walking

This is the practice Cal Newport, like many before him, has adopted and recommends. Want to ponder? Take a walk.

Rhythmic movement

Running, yoga, hiking, dance, rowing or cycling are examples of meditation in motion. As you engage in them, notice when your thinking drags you elsewhere and return your attention to your physical experience.

Meditation

You can practice meditation while sitting still, moving, doing breathing exercises, listening to music or drumming, chanting, following words spoken by a guide, or some combination of these.

Discourse

Talking to trusted people about your experiences helps you become more aware of your experiences. Discussion can be used as a deep self awareness practice

Making art

When we get immersed in a creative process, you are usually, also, listening closely to yourself. You both act on your impulses and decide not to act on them. When art is used as a self awareness practice (rather than strictly a crafting exercise) it can be a powerful tool. We can make models and code as we would make art.

There is no right way to cultivate self awareness (though I'm personally a big fan of writing.) Trust yourself to discover the ones that work for you. There are many teachers whose purpose is to support self awareness practice, you might find that having one helps you.

When done with the right mindset, *reading* can be a foundational and valuable practice for understanding your own thinking. When Joseph Campbell, who wrote *The Power of Myth*, was asked what form of meditation he practiced, he said “I underline sentences.”

I believe that reading and writing are the most nourishing forms of meditation anyone has so far found. By reading the writings of the most interesting minds in history, we meditate with our own minds and theirs as well. This to me is a miracle.

—Kurt Vonnegut

The practice that you choose is not what matters most. What matters most is that you show up for it.

Everything in our Blindspots

One day when I was twenty-one years old, I was browsing the 294.x section of the Penrose Public Library in Colorado Springs, Colorado. Feeling bedraggled and lackluster, I wandered along a dimly lit aisle, dragging my fingers along a dusty shelf. I'm not looking for anything in particular, wandering a subject area (Religion) I know nothing about beyond my Catholic upbringing.

An interesting book catches my eye, I scooch it off the shelf. The book is about Buddhism.

Reading the introduction opens my mind like entering Narnia through the wardrobe. I did not suddenly find faith. I found numbers: 70% or more of people in the world practiced something other than Christianity. I'd been raised in a dualistic world with two options: Christian or Jewish. I'd always struggled with that duality and now, I realized, it wasn't a duality at all.

To be clear, if you'd asked me that morning, "Do you know there are other world religions, Diana?" I would have said "Yes, of course, I do!" (I knew everything at 21.) But I'd hadn't yet experienced how massive and diverse the world was. Clearly, I understood, there were major gaps in my knowledge. The magnitude of how much I had to learn stared me in the face.

My lack of wordliness might not make sense if you didn't grow up in the Boston area before the internet. We believed ourselves to be the center of the world. History, as I was taught it to me, was all about New England. Not simply "American" history. History history, in one straight line from the Ancient Greeks to Beacon Hill.

Even in the increasingly interconnected world of today, we all live in binary mental models. We all have big blindspots, realities that exist outside of our current definitions of "reality." Even when we don't agree with the thinking around us, our concepts are constructed by it. We will live in those constructed dualities and limited worldviews unless or until we proactively practice seeing beyond them.

You don't see something until you have the right metaphor to let you perceive it.

—James Gleick, *Chaos: Making a New Science*

Peering into that library book, a crack opened in my mind, letting in the thought: *everything I've been conditioned to think may not be true.* My fundamental mental models are flawed, not because they are a lie but because they are limiting. Insufficient. And inflexible. As I've matured, ideas which

seemed correct (enough) in one context were not correct in another.

Paradigms shift.

I need to make up my own mind. There is no system thinking without that skill. We all do the work of making up our own minds. We do that work together, integrating and synthesizing views. We construct our conclusions and share our reasoning so others can do the same. We step right into ambiguity and wrestle with it.

To think in systems, you *proactively seek out what you don't know you don't know.* Sounds like a paradox, doesn't it? How do you know what you don't know? You ask yourself "what else might be true?" When face with two choices, you ask yourself "might there be alternatives I haven't considered?" When feeling pressed into action, you ask yourself "what if I do nothing." There are a myriad of ways to look for your blindspots, and in systems, most solutions will be waiting for us there.

We are all, in some ways, like the young woman from Boston, someone arguing against other people's ideas (while sippin' Dunkin Donuts coffee), against the structures of the most

powerful ideas, forgetting to think beyond the obvious. Small shifts in our thinking practices, coupled with self awareness, can open doors to a vast world of alternatives.

I still have Narnia moments, when a book or conversation reveals the limitations of my own mind. In systems thinking, we settle into not knowing. We have much to learn on the journey in front of us. Everything we know now is defeasible.

And that's okay.

Decision Making is a Noisy Process

We tend to run our whole life trying to avoid all that hurts or displeases us, noticing the objects, people, or situations that we think will give us pain or pleasure, avoiding one and pursuing the other.

—Charlotte Joko Beck, *Everyday Zen: Love and Work*

Systems thinking is not about gaining more control ... it is about making better choices. Finding leverage points, places where a change will have a big impact. “We can’t control systems or figure them out. But we can dance with them!” (Meadows)

Learning this dance begins with becoming aware of how we come to conclusions and make decisions. It's also observing the ways we avoid making decisions. What happens when you make choices? What drives you? You will likely discover that your own process mirrors many choice-making processes around you.

When I see chocolate, I want to eat chocolate. Chocolate has dairy and sugar, both make me feel ill. I might want chocolate but there is no universe in which I *need* chocolate.

Yet, I'm conditioned to feel that chocolate *matters*. At a conference recently, the hotel put out gluten-free chocolate brownies for a snack. I ate one. Then went back for another. Then got one for a friend I thought would enjoy a brownie. But I ate it instead. All after, I thought about chocolate brownies, a constant circuitous inner debate: "Do I eat the chocolate or don't I eat chocolate?" Some examples of my mindstates:

- "I deserve some chocolate!"
- I feel suddenly certain that I'll *starve* if I don't eat some.
- "Don't eat it, I'm getting fat!" (I have not gained any weight, this is my mind bullying me into not eating it. This usually fails.)

- “Have I been “good enough” to “cheat” with chocolate?”
(Junk food isn’t a well-deserved “cheat” - it’s just junk food.)
- “I’m on vacation so of course I will eat chocolate!” (Why does sugar equal fun? Also, I’m not on vacation.)
- Maybe it is a gluten-, dairy-, sugar-free chocolate brownie that tastes like a brownie. (Examines ingredients for magic).
- I ask others if they want a brownie. If someone else wants one, the social cues somehow dictate that I should eat it.

Keep in mind, there is no “decision” required. I don’t need to eat a brownie. There’s just a lot of noise in my mind.

My confusion wasn’t really about chocolate brownies, it was about competing priorities. I want short-term pleasure and longer-term health. My circuitous thinking tries to prioritize both things even though they are naturally in conflict. Our minds go to great lengths to *avoid making tradeoffs*.

Our technology lives are full of chocolate. Self awareness practices help develop our objective interpretation. I can tell you the brownie story because of self awareness. I observe my thinking, feeling and experiences often enough to see them as they arise.

How many times a day do these dramas play out in our daily work lives? Many. Going to great lengths to avoid making tradeoffs is so common, it describes every codebase I've seen. I call this "carboat". One team wants a car. Another team wants a boat. The engineers build a carboat, which nobody wants.

Noisy, conflicting opinions about what is best to do can feel like walking down the middle of a road during a hurricane. We get blown around by every idea, veering off course and getting lost in the fog. Distinguishing which choices are under our control and which are not is also difficult. (Am I controlling the brownie decision or is it controlling me?) Systemic forces push against some of our choices and encourage others, using feedback loops. This entire book is about working inside of those forces but your ability to do so is limited by your self awareness.

Chaos, the book by James Gleick, tells how the emerging science of systems thinking began ... with a meteorologist literally studying weather patterns. That makes so much metaphoric sense. Making matterful decisions in the midst of circumstances I cannot control ... this is life in systems.

Things Self-awareness Taught Me

To encourage your practice, here are some insights that self awareness practice has revealed to me over the years. Perhaps you'll make a list of your own, to help encourage your practice?

About my thinking process

- I'm not nearly as in control of my thinking as I think I am. (This is a recursive insight.)
- My thoughts are not always truthful, reasonable, intelligent or have my best interest at heart.
- My first instinct about whether or not to trust my own thinking, before objectively considering it, is unreliable.
- The quality of my thinking improves when I'm focused. It degrades when I get busy.
- I am constantly trying to work against uncertainty rather than roll with it.
- Complexity is a nourishing pool that I can dive into. Generally, I've preferred to stand on the edge and shout at it.

About my social interaction process

- My thoughts about other people are not always truthful, reasonable, intelligent or have their best

interest at heart.

- I think a lot about what I woulda coulda shoulda said or what I might say later, whenever I don't know what to say.
- I am more swayed by other people's opinions more often than I want to admit.
- Change is uncomfortable.
- I don't really know what I think until I try to write it down (or model or structure it in some way).

About the way emotions interact with thoughts

- My emotions are unreliable. My emotions also express valuable information that I ignore at my peril.
- I often make bad decisions when my emotions are leading the way. I make even worse decisions when my emotions are ignored.
- Mental models that are emotionally familiar but functionally incorrect are difficult to change.
- What I perceive as my own needs are often manufactured needs. For example, “relaxing” activities, like watching television, aren't necessarily relaxing. Television creates a “need” for more television.

About my professional patterns and processes

- The quality of my deliverables are equivalent to how well I can discern between good thinking and habituated thinking.
- The level of psychological safety in my environment is directly correlated to the quality of work I can deliver.
- When someone engages with my work in a way that nourishes and strengthens it, I want to do more (and better) work.
- Being bullied can feel incredibly isolating in an environment that accepts it as normal.
- I overcomplicate things until I can see the elegant simplicity.
- Not everyone's thinking is equally worthy of consideration. (This is blaspheme in some cultures I've worked in. Yet, I maintain that sharing opinions is not the same as thinking.)
- Fighting other people's thinking is usually a waste of energy. Demonstrating sound thinking more often gets me what I need.

About the world around me

- Many information systems are trying to control my thinking and behavior. (Like advertising.) Sometimes it works.
- An underlying mental model in the social system around me is that something is wrong with me and someone else knows how to fix it. (My face is aging but that cream will fix it.)
- The mental models that have shape how I think are often contradictory and sometimes, nonsensical. Discerning which models to rely on and which to discard is challenging.
- Whenever a single root cause of a difficult systemic problems is offered, it doesn't work. Composting won't end global warming and Keto won't cure cancer. I know this and still, I look for a single root cause to my discomforts.
- Authority is no guarantee of wisdom or truthfulness. Neither is anti-authority.
- Very few things are dualistic, right or wrong, Left or Right, Drupal or Wordpress.
- We are what we are aware of.

Early in my career, I imagined myself a knight charging into initiatives with her sword drawn, tirelessly working to bring

new technology and ideas into the world. I believed (erroneously) that I efforted things into existence.

Sure, effort is always required. But what kind of effort? New ideas grow faster when I nourish them with my full attention. I'm not great at nourishing. Okay, I am truly terrible at slowing down and nourishing thinking. Everything in subsequent chapters, and systems thinking as a whole, depends on making time and space for thinking. Then caring for that thinking, even when caring means pulling the weeds in your own mind.

Understanding my own mental and emotional patterns has made me a more nuanced systems designer. When I'm caught up in the noise, I recognize that sooner. I've been burned out and angry, after investing myself in a situation that doesn't change. I wasn't wrong but my frustration, blame and negative reactions eclipsed new thinking. Through self awareness, we can slowly but surely learn to be the change we want to see in the world.

Summary

We begin learning systems thinking by becoming aware of our most intimate and familiar system – ourselves. We practice

metacognition, awareness of our thoughts and how we decide. And self awareness, understanding thinking as a holistic experience that includes our feelings, experiences and physical sensations.

We cannot improve our thinking unless we are aware of our thinking. So dive right in and invest time every morning writing down your thinking.

Our thinking is limited by all the things we do not see. Systems thinking is expanding our view and learning to make up our own minds. We take action based on what we know and we accept that later, we might make different choices.

Self awareness enables to continuous improve our choices. We are always navigating a storm of uncertainty and competitive cognitive forces. That's okay.

Self awareness has been my greatest teacher. It has shown me that my thinking follows patterns, like avoiding making tradeoffs (a pattern that exists in most technology initiatives.) Many of the patterns that exist in my own mind exist in the organizations around me.

Practices

Every morning, I get up before dawn, grind coffee beans and make a half-caf double shot of espresso. I add Silk Original Creamer and carry my cup to my study, where I sit in my lounge chair. Leaning back, I pull my blanket over my lap and the cat curls up next to me. I click my Pilot Vanishing Point fountain pen (the same one Neil Gaiman uses) and open my A4 Firma-Flex notebook.

“Alexa, Good Morning.”

“Good morning, Diana.” She tells me the date, time, weather and sets a timer for an hour.

This is not self discipline. I wish I were that macho. I need this time. I don’t know, anymore, who I’d be or how I’d think without my morning practice. Students in my courses have reported similar results. I rarely miss a day, even if it means writing at 3:30 am before I leave for a flight. The silence before dawn is now my favorite time of day. Pairing the practice with a delicious cup of coffee, made just the way I like it, helps get me out of bed.

In my journal, I make lists. I ask myself questions like “if you knew you were going to die in a year, what would you do differently?” I write letters I don’t intend to send, just to hear

how I feel. I think through strategies, choices, options. I prioritize. I agonize. I get silly; I face my fears. What I write depends on the day, though there's a lot of repetition. I complain about the same things, again and again and again, until I realize that I'm stuck. I use my writing practice to begin thinking about them differently.

If I were forced to recommend one and only one self-awareness practice I'd pick this one. But I've known people, my husband included, who haven't benefited from it the way I do. So I temper my recommendation: start with this practice. Do it for a month, or three months. See what happens. If it takes root, continue. If not, try other practices. (Do give it time to take root before deciding though.)

The instructions are simple. *Choose your constraint (3 pages, 20 minutes) and keep your hand moving.* That's it! If you don't know what to write, write "I don't know what to write" until something else occurs to you or you reach the end. This is a movement practice. In our daily lives, we are constantly interrupting and interpreting and judging our thoughts and feelings before we even allow ourselves to be aware of them. Flow with them, let this practice help you become aware.

In case you get stuck, here are some prompts:

- I love
- I hate
- I am disgusted by
- I want
- I remember
- I wonder
- If I quit my job today, I would most like to
- If I left [situation], I would most like to
- If money were no object, I would
- If I could leave one thing in the world when I die, it would be
- If my parents were [different people in a specific way], I wouldn't be
- If my parents were [different people in a specific way], I would be
- [Someone] is absolutely wrong about
- When I was little, I wanted to
- When I am old, I want to
- In the meeting yesterday [someone] said [something] and I think
- I think
- I feel
- I need
- Without [someone] in this situation, we could

- If I could change one thing about [situation], it would be
- My favorite book is [book] because
- If three millions dollars showed up in my bank account today, I would
- If wish I invented
- I wish I could invent
- I help others the most when I
- I hurt others the most when I
- Something I'll never recover from is
- I need to forgive myself for
- I want to recover my
- My most treasure professional skill is [skill] because
- My biggest professional regret is
- If I woke up tomorrow with the role of my dreams, it would be
- I am grateful for
- I want to change
- I fear
- I trust
- The thing I'm good at that nobody sees is
- The strength I wish was trusted and recognized is
- The weakness I wish I didn't have is
- I can make a difference by
- I teach

- I learn
- I am so hard on myself because
- I am so hard on [someone] because
- If I woke up tomorrow and one thing had changed, I'd like that one thing to be ... and my new life would be ...

Questions for Your Journal

- What do I need in order to create a daily practice of self awareness?
- What are my blockers?
- How will improving self awareness improve my work as a technology professional?
- What are three examples of recent experiences where my self awareness helped me?
- Have you had experience when someone else's lack of self awareness made things difficult?

Chapter 5. Replace Reacting with Responding

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 5th chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at sevans@oreilly.com.

The difference between Action and Re-action is originality and timing.

—Aniekee Tochukwu Ezekiel

As you practice self awareness (discussed in the previous chapter), you'll notice that you are constantly reacting to your experiences. Some of your reactions are benign, like putting on a sweater when you are chilly. Some are explosive, like screaming at a driver who cuts you off. Most of your reactions exist somewhere in between, a constant shifting of our mental and emotional state depending on inputs. We might not realize how often we are controlled by these reactions.

An engineering colleague once told me that “feelings have nothing to do with my work.” This, in my experience, could not be further from the truth. We are *always* in the midst of reactions, our own and everyone’s around us. Ignoring them is ineffective.

The more someone isn’t aware of their reactions, the more emotional labor they are likely to require from others. When someone is reactive, they make a lot of mess and a lot of noise. Perhaps you’ve seen this pattern? A suggestion is made, people get upset and, through backchanneling, a chain of reactions starts?

This isn’t a chapter on emotional intelligence, though I do encourage all technology professionals to explore it. As a group, we experience a wide range of ability to recognize and label

emotions (in ourselves and in each other). That's okay, noticing our reactions is about noticing *that* we feel – not being quickly able to describe *what* we feel. This is a book about thinking and reasoning — our thinking is intertwined with our mental, emotional and physical reactions. We will focus on noticing those reactions, because more often than not, they block our ability to think systematically.

[Figure 5-1](#) is a slightly different version of the iceberg model, where the top levels show visible events and trends. The iceberg process is happening all around us but also inside of us. We react to situations in ways that are informed by our mental models and reinforcing structures. We may or may not be aware of those underlying layers. But they will inform the things we say and do.

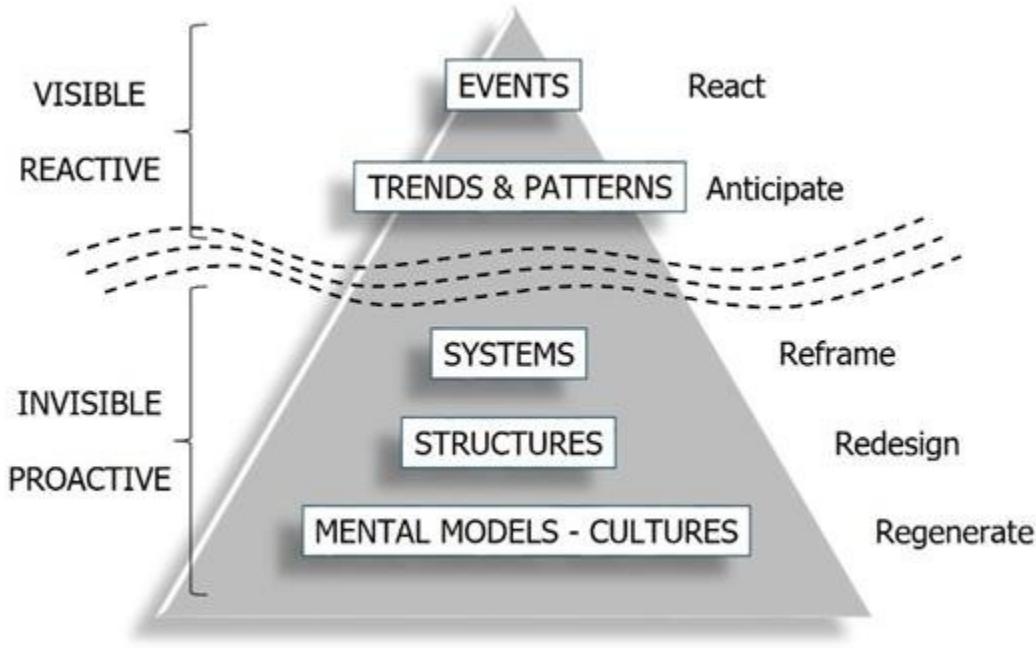


Figure 5-1. by Bryan, B.; Goodman, M.; Schaveling, J. Systeemdenken; Academic Service: Cambridge, MA, USA, 2006

And we will usually believe our reactions are caused by circumstances outside of us, rather than our underlying system. (Which is only sometimes true.)

In Part 3, we will develop reasoned responses, structuring our systems thinking to test and share. Before we can do that work, we need to make space for our reactions.

In This Chapter

You will learn that noticing your reactions helps you think and communicate more effectively. We will usually make a problem worse by reacting. We can instead create judgment-free

awareness of our reactions and use them as an information source. In order to provide systems leadership, we need to stop sharing our opinions and instead, construct sound reasoning. The first step is to become aware of your own reactive patterns. The concepts we will explore include:

- Noticing and creating space for your reactions.
- Understanding that opinions are less helpful than carefully-considered systems thinking.
- Empathy is important in systems thinking but does not require ignoring your own experiences.
- Choosing to practice alternatives to reactive thinking is a great first step towards strengthening your mind's ability to navigate circumstances.

The practices you'll discover are:

- Cultivating the “yes, and ...” habit.
- Practices that settle down reactions rather than fuel them.

Noticing Your Reactions

I'd like us to begin by pausing for a moment. Remember a meeting, or other discussion, where someone said something you didn't like. Remember how it felt, the thoughts that went

through your mind, what you experienced afterwards. What do you remember?

I'm remembering an exchange that happened in a group chat. I responded to a post in a way that I intended to be respectful, acknowledging and friendly. The original poster did not take my response as I'd intended it. I could say that, for about 10 minutes, the whole thing blew up.

Fortunately, I had an appointment and could not respond. Instead, I watched a parade of reactions happen in me:

- I felt sorry for inadvertently hurting their feelings and wanted to immediately write an apology.
- Simultaneously, I felt defensive, and misunderstood – I was mentally justifying what I said. (This is the most common reaction, the urge to push back and explain.)
- I was confused about what to say and do. My mind kept replaying the exchange and coming up with new things to say. Except they were mostly the same things.
- I felt guilty for not knowing the right way to respond.
- I felt guilty for not believing it was my fault and not being sorry.
- I felt sad that the person did not trust my intention and give me the benefit of the doubt. I thought we were closer than

that.

- I remembered a difficult exchange I'd had with this person in the past and decided this was what they "always do".
- I felt responsible because I was rushing to my appointment and hadn't taken the time to fully understand what had been shared before I chimed in.
- I resented having to deal with this issue.
- I was mad at myself for checking my phone.
- I felt bullied and embarrassed.
- I felt frustrated when the original poster deleted some responses, so the notifications I received didn't match what other people, joining after, would see.
- I decided to quit the whole situation. Multiple times.
- I decided to practice all the things you will read in this chapter. Because this happened *while I was writing it.*

I had to practice the recommendations you'll read in this chapter because, in the heat of things, we all need to practice them. The outcome led me to three points that are so essential to good systems thinking, I encourage you to read them twice:

1. I would have made the actual problem worse by reacting.

Notices that none of my reactions had anything to do with the actual problem we were trying to solve. You don't even know, reading my reactions, what the actual

problem was. By the time I responded, that problem was resolved and everyone was happy.

2. Had I reacted, I would have done more damage to the relationship.

The next day, we had a clear, friendly, respectful, quick and clarifying exchange that resolved our miscommunication. I understood how I contributed and learned from it comfortably.

We understood each other better as a result. We built trust.

3. My reactive patterns are so predictable, half the items on the list above would be there in every example I shared.

Even though my reactive patterns rarely get me what I actually need, I always believe they are trustworthy in the heat of the moment.

Reactions are recursive. We have a reaction and we react to our reaction and react to our reaction to the reaction ... this is a highly-combustible process. With self awareness, we notice the heat generated by our reactions. The practices in this chapter help us experience them without setting the situation on fire.

Create Space for Your Reactions

Our reactions to other people's thinking or our circumstances, like all systemic phenomena, depend on many factors.

Understanding the root cause takes time and consideration. Our goal is to give ourselves that time. There are four ways we are practicing when working with reactions:

- Notice them without judging them.
- Look for systemic information in our reactions.
- Don't suppress them.
- Manage your own reactions and let others do the same.

When you feel upset by something someone said in a meeting, then you judge yourself for feeling upset ("you are too sensitive"), now you have two reactions to deal with. Your reaction and your reaction to your reaction. The judgment arises from a desire to fix yourself, or someone else, like the person who upset you. Experiencing emotions and thoughts that may or may not be out of sync with our circumstances is an inevitably human experience. It doesn't mean you are broken, it just means you are experiencing a reaction.

Our reactions are messy and raw. When we create inner space for them to arrive and be a little rowdy, we can help them settle

down and become orderly. If conceptual integrity and cohesion are the most important considerations in systems design, then reacting is the storm that is constantly pulling those qualities apart.

NOTE

Our goal is to increase our capacity for holding reactive thoughts in our state box, while we construct more-cohesive output.

We all experience table-flip moments, an urge to push back on the person disagreeing with us in a meeting, our partner, our boss. When we do push back, we might feel better in the moment but we rarely say something that improves the situation. When a group of people are stuck in reactions, they create momentum that moves them further and further away from a cohesive conclusion.

An important caveat: Sometimes, your urge to push back is the correct response. “No, stop, back off, this is not okay.” How do you discern? There is, alas, no rulebook to follow. It depends. We get to learn through experience. Linear thinking sorts behaviors into “right” and “wrong” – reacting is not (necessarily) either.

Reacting is, when it comes to systems thinking, potentially unskillful. You are being derailed by other people's thinking. When we practice, we create more options for ourselves, including the option to react in the moment.

Here's an example: a new architect is meeting with an engineering team for the first time. Five minutes into the meeting, an engineer shares his screen to demonstrate a problem. A Slack notification pops up, sent to by the team's boss, who is also in the meeting. The notification says "I don't think we should trust what she [the architect] says, she's not technical enough." The architect, of course, sees this. So many reactions arise!

A month later, that same boss is quite happy with the architect. But that's not the point of the story. The point of the story is that the actual problem, being shown by the engineer, would not have been solved if the architect had been derailed by reactions. The boss was wrong. The architect's technical skill became apparent over time. I'm sure you can imagine, though, how challenging it was for the architect to continue focusing on the problem.

In this situation, the architect did not respond to the comment. In a different situation, she might have paused the meeting and

dealt with it. There wasn't a "right" decision here. But there was a reactive decision.

Our reactions are usually defensive, which makes them unreliable. They derail productive insight because the focus shifts to resolving the inner discomfort. Our reactive thoughts are filled with emotional baggage, cognitive biases, conditioned responses, misunderstandings and false narratives.

Technologists are especially prone to reacting strongly whenever someone is wrong (but not when someone is right.) Logical fallacies, which we explore in Part 3, thrive in our reactive minds.

That's okay! Our reactions are still worthy of our attention. The architect felt insulted and that's a fair response. This wasn't the first time strangers have presumed that the architect isn't technically skillful or capable of leadership. When we are reacting to a pattern, that pattern probably exists as a systemic feedback loop. Reactions contain information, insights that are as-yet unformed, that might help you construct a response. The salient point isn't that our reactions are bad, it is that they might be a tangled mess that lacks cohesion.

After the meeting, the architect was like someone looking at stuff piled up in her garage. She needs to tidy up, sort out which

thoughts to keep and recycle the rest. In systems thinking, we are tidying all the time. Decluttering and prioritizing communication. Organizational thinking is more likely to set things on fire than build something. But when we speak for the system, we try to create elegant simplicity from the mess. We don't control and demand; we don't overly simplify or discard complexity, we dance with it.

We don't necessarily act on our reactions ... we also don't suppress them.

I have a chronic habit of suppressing what I think and feel when I'm in upsetting situations. I clamp down, hold my breath, put on my most-reasonable "adulting" face. I *manage* these situations. I learned this skill during chaotic and threatening childhood experiences but most of us have been in situations that pushed us to suppress.

Managing reactions is sometimes a good trait to have professionally. It helps us avoid devolving into unproductive squabbling. But it isn't authentic. When I stuff down my feelings, controlling my body language and tone of voice, silencing my reactive thoughts ... they all come out later. Except later, they come out in the form of resentment, confusion, exhaustion, fighting about nothing with my husband, pervasive

frustration with my job and eventual burn out. My reactions pile up until I perpetually feel unheard and unacknowledged.

Those feelings have been an accurate assessment of some situations, for sure. But they also indicate that I am also not listening to and acknowledging my authentic experiences.

Reacting to our reactions by suppressing them is a tough habit to break. Social feedback loops reinforce that with pressure on people, especially girls, to “be nice.” Nice is good! But hiding your thinking is not good.

The flipside of this habit is managing other people’s feelings. I have seen this trend accelerate in tech groups, putting managers and organizers in the role of “cat herder.” Their unacknowledged job is to keep people happy and sort out all the reactions, trying to keep people on track.

Managing other people’s reactions is the work we do with toddlers. Leadership is the practice of creating psychologically safety not by managing people’s reactions but by expecting everyone to develop self awareness, do their own emotional labor and pay attention to their impact on others. Constructing an environment driven by productive and intelligent responses

includes respecting that people need to think and consider things deeply. Sometimes they need some space to do that.

More importantly, leadership is developing a wise relationship with your own reactions.

People who've gotten used to me "managing" situations, doing the emotional labor, react negatively when I stop. I've found it challenging to tolerate my discomfort as I move towards experiencing the fullness of my authentic thoughts and experiences. Sometimes, not controlling situations but instead, taking care of my thinking, has felt like quitting smoking or giving up sugar or denying any habit that self soothed but otherwise delayed experiencing what was feeling.

The practices I will recommend are decidedly non-technical. Breath. Write. Walk. Wait. That doesn't mean they aren't critical to quality technical design. In your experience, do groups embroiled in drama, in need of cat herding, deliver quality code? In mine, it is the groups who know that nobody gets to be a cat that deliver great work. The more we practice, the more impactful we can make our (eventual) responses.

Opinion-driven as Normal

Technology culture is, in my experience, opinion driven. We are quick to share our views or judgments about something, whether or not we have sound reasoning to support our thinking. Too often, we approach idea sharing with a “This is my thought. Change my mind.” approach.

Whenever two or more are gathered, opinions about technology proliferate. We form tribes with those who share them.

I love our opinions. I’m not a big socializer. At 9 pm, I’m not heading out to dinner, a bar or a party. I’m heading to bed, carrying my Kobo.

Except at tech conferences. There, I’m wide awake, long past my toddler bedtime. I once left a conference after-party in Bucharest just in time to catch my early-morning flight. At most conferences, I’m among my tribe, endlessly discussing ideas.

We are attached to our strong opinions, in some ways, they define us. While it can be fun to debate opinions, when it comes to systems thinking, being opinion-driven has its downsides.

We debate dualities.

Our opinions are usually structured as “for or against”, “yes or no”, “right or wrong”, “option A vs option B”. Nuanced, intersectional nonlinear thinking styles are uncommon and sometimes unwelcome. I’ve rarely heard “for or against Agile” discussed as “it depends on the circumstances and how it’s implemented.”

We want to be right.

We love our opinions and we fight to defend them. We want to be right and we want people to agree that we are right. This is antithetical to systems thinking where we want to learn and grow.

We ignore context.

Very few things in life, and in tech, are Always Right or Always Wrong. Most things depend on the context. What worked for Netflix or Spotify might be a disaster somewhere else. Opinion-driven discussions don’t usually focus on understanding the complexity inherent in each context. It depends. Figuring out what “it depends” on would serve us better than debating a universal truth.

We focus on “wrong”.

When someone is accused of being wrong, even the most emotionally intelligent people will get defensive.

—Karen Kwong

When I share an idea with a group of people who don't work in tech, and their responses are kind, respectful and curious, it surprises me. I've learned to anticipate reactions that immediately point out what I said wrong. Changing this habit is the single most impactful practice I've adopted.

We are emotional.

For a culture that prides itself on Spock-like science logic, people sure do get incensed about a lot of things. I hear endless disdain for other people's thinking, especially people who don't code. I've seen, grudges endure and backchanneling run rampant.

I'm fairly sure Spock wouldn't be gossiping in Slack about ideas he doesn't like.

Generally speaking, there is no avoiding forming opinions. When practicing systems thinking, it helps to remember that the definition of opinion is "a view or judgment formed about

something, not necessarily based on fact or knowledge.” There is more work still to be done.

We are knowledge workers. Our expertise is not demonstrated by our “correct” opinions. Our expertise is demonstrated by our ability to change, drop and transform our opinions as we learn and grow.

The Stories Don’t Have to be the Same

I’m standing on a street corner, waiting for my husband to pick me up at our agreed-upon time. Cool spring rain is pouring down and I don’t have an umbrella. My light jacket is soaking through as the minutes pass beyond the pickup time.

I scowl at each car that isn’t our car. I’ve called and texted him multiple times. “Where are you?!” “I’ll be late for my meeting.” “My laptop might be ruined.” “Where are you? My lunch is getting cold.” This sucks. I’m growing increasingly angry as the chill sets in.

Then fear takes over. He hasn’t answered. What’s happening? Maybe he’s dead?!?!

14 minutes later, my husband pulls up. When I see the car, my fear evaporates. Okay, he’s not dead. Now I’m angry again, the

words already forming on my lips as I open the door.

Earlier in our relationship, an escalating Reaction Fest would begin. He'd react to my anger because, from his point of view, it was unfair or unwarranted. He'd react to my reaction. I'd react to his reaction and we'd spiral all afternoon.

Today, he says, "Oh my goodness sweetie, you're so wet, I'm so sorry, that must have sucked." He listens to my frustration. He digs around the backseat for a towel to dry my hair. He asks me if I still have time to make it to my meeting. "This is awful and sucky, you must have been so worried."

Then, as I take a breath, he tells me his story.

He'd stopped at the grocery store on his way to meet me, as agreed. When he arrived, he reached into his pocket to check the shopping list on his phone. His phone wasn't there. We'd visited three other places that day. He'll need to retrace his steps. He decides to quickly run in and buy groceries, pick me up for my meeting, then go find his phone.

Twelve minutes later, groceries paid for, he rushes out to our car with just enough time to get to me. Another car is stopped behind ours, blocking him in. The driver's side door is open; there is no driver inside.

My husband stomps around the parking lot for a minute, looking for the offending wanderer. He yells, “Hey!” but no one is around. He goes back into the store and waits impatiently in line at the customer service desk, behind someone returning apples. When the apple person moves off, he asks the service representative to page the driver.

“Oh!” she says, “he was having chest pains. He came staggering in, asking us to call an ambulance. The EMTs are on their way. I’ll get someone to move the car for you, if the keys are in it.”

He goes back outside and discovers that the car parked in the opposing spot has left. He pulls through and drives as quickly as he can, while still being safe, worrying about me standing in the rain.

In the past, when I discovered that my feelings of anger, frustration or confusion were unwarranted, I’d feel guilty for having those feelings. I’d suppress them. Why couldn’t I stay all Zen while waiting in the rain with my ruined lunch and no ability to get in touch with him?

What changed our dynamics was one little mantra: the stories don’t have to be the same.

We learned to empathize with the experience the other person actually had ... not the experience they would have had if they'd known things they didn't know. Not the experience we think they should be having. I don't have to make him wrong, or make myself wrong. The whole situation created frustration for everyone. Even though nobody, including the driver of the blocking car, is to blame.

When we are reacting, we are almost always blaming. We are always projecting our story onto other people. When we don't understand what's going on, blame tries to fill the gaps. Sometimes, blame is accurate. But it will rarely help us *solve* the problem.

Systems thinking is integrating the story of standing in the rain with the story of being stuck at the grocery store with no phone. It is integrating the business point of view with the technological reality. It is understanding the pain other teams experience when we make upstream systems changes. It is recognizing that your point of view, real and matterful as it is, is not the whole story.

A confluence of events happened that day, not a recurring systemic issues that needed addressing. My husband is rarely late. Blame will push you to fix fix fix. Sometimes that's accurate

and something there's no problem to fix! Sometimes, the right move is to simply dry off and go home.

Systems thinking depends on wise empathy, understanding other points of view, “seeing” the circumstance in a more-holistic way. There is a limit, though, to how much understanding and the practice of empathy can improve a situation.

The Tyranny of Understanding

Understanding other people’s thinking, experience or behavior doesn’t mean you have to accept or go along with it. You can empathize and still ask for a behavior change. If my husband regularly left me standing in the rain, his story would have been understandable but the pattern still needs to be changed. Understanding is not the work that needs doing.

Identifying, and communicating, causes of systemic antipatterns and blockages isn’t the same as blame. Blame, as I’m using it here, places responsibility where it doesn’t necessarily belong as a reaction to circumstances. A person who, for example, robs a convenience store isn’t being blamed

for the robbery, they are being held responsible and accountable for it.

Here's another example: Engineers on another team dismiss my recommendation, giving a myriad of reasons. They share their feedback in a derisive tone but that's not the primary problem. The primary problem is they dismissed a critical architectural pattern and industry best practice that needs to be implemented before we can move forward.

I share the feedback with the engineering leader, asking them to help me create shared understanding of this emerging software pattern. How can we bring the team up to speed? Their response was, "You need to understand them, Diana, this technology change is an existential threat to them." In essence, they took an "everybody is entitled to their opinion" view, as if the recommendation and the opinions shared about it were equivalent. As if understanding, not training, was the work needed.

This is a common place to get stuck. Sometimes, more empathy and understanding is not the solution, especially if the empathy and understanding reinforce negative sociotechnical behavior. Action and accountability, learning and collective reasoning,

matter too. This often means acknowledging a difficult or politically-challenging reality.

When you run into this type of roadblock, you might need to find another way around it. Leadership, in systems, includes transforming sociotechnical thinking and behavior patterns that hold a system back.

The Practices

How do we minimize the amount of reaction we contribute? What do we do when reactions get triggered? How do we create space?

First, and most importantly, we notice when we are reacting! We watch our thinking and feelings, noticing the warning signs of a reactive pattern. You'll see that your reactive patterns are predictable and maybe even tedious.

Here are some practices that might help. The first one, when thinking in systems, is essential.

“Yes, and ...”

This practice is the easiest and most impactful way to generate the right mindset for thinking together. “Yes, and ...” is a rule followed by improvisational comedy actors. These actors get up on stage in front of a live audience and make up scenes in real time. This is daunting work. The television show “Whose Line is it Anyway?” is an example of improv.



Figure 5-2. https://en.wikipedia.org/wiki/Improvisational_theatre

“Yes, and...” is critical to their work because it trains them to cooperate, to think well together and create a flow of ideas, rather than shut down contributions before they have a chance to flourish. Improv teams warm up playing “Yes, and ...” the way runners stretch before a run.

The practice is simple: when someone expresses an idea, you acknowledge it and add something to it. In your body and mind, you accept what is happening, you mentally say “yes”. When you respond, you add to the idea, flow with it, strengthen it.

To be clear, this does not mean agreeing! Acknowledging has nothing to do with agreeing. Here are two examples:

Hollis: We need the API response times to be faster because too many client connections are dropping after 3 seconds.

Briar: The response times are fast enough. That’s not the problem.

OR

Hollis: We need the API response times to be faster because too many client connections are dropping after 3 seconds.

Briar: Thanks, Hollis, for describing why we need the responses to be faster than three seconds. I wasn’t aware of slow response times. What alerted you to this problem?

You are always listening critically. You are working together to generate momentum. As soon as someone contracts into “No”, the whole scene crashes. I’ve experienced this as an actor – it is

nearly-impossible to recover from and the audience can't help but notice. I've experienced it in countless tech meetings too.

Perhaps you can already see why changing this one habit would improve most technology discussions? In my experience, technology culture is a “No!” culture. Our habit of only responding to the thing we hear that is “wrong” is so pervasive, XKCD’s fifth most-popular comic, [Duty Calls](#), describes it.

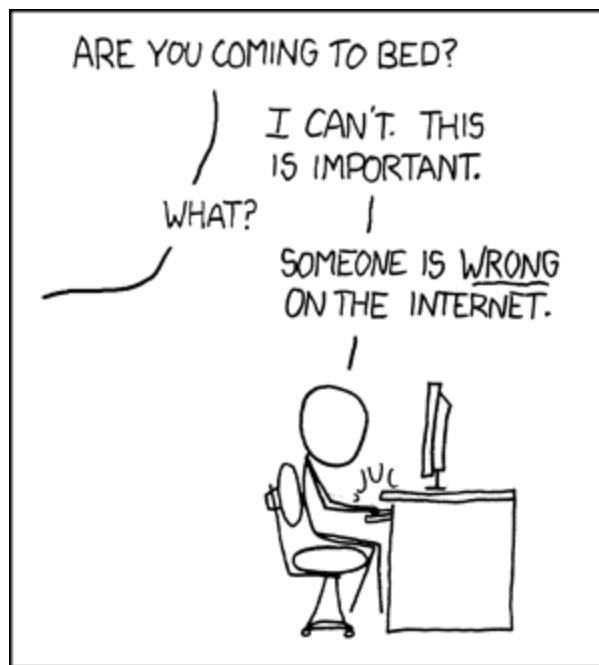


Figure 5-3. From xkcd, a webcomic: https://imgs.xkcd.com/comics/duty_calls.png

“Yes, and ...” is a respectful approach to discussing ideas: Acknowledging what the previous person said and adding to it.

“Yes, and ...” isn’t about everybody agreeing or pretending to get along. It is about breaking the powerful mental habit of

contracting, demanding others “change my mind”. It is an agreement to do the necessary systems-thinking work by leaning in and thinking critically, together.

When thinking in systems, knowledge workers are technology’s improv teams.

The 24-hour Rule

This practice changed my life. I was caught up in a situation that was full of demanding drama. In the midst, I was given this advice: Wait 24 hours to respond. When I received the next emotional email demanding an answer I did not respond for 24 hours.

This practice didn’t (at first) change the emails I received. It changed the answers I sent (when and if I sent them). I realized that I wasn’t giving myself space and 24 hours made my responses wiser.

Professionally, I’ve been surprised by this rule. When I waited 24 hours then read the email again, at least half time ... I’d simply misunderstood.

Breathe

The first time I taught breathing exercises to a group of software engineers, I feared they'd rage quit the workshop. Instead, they asked me for more. Stress is a constant part of our work, breathing exercises actually help.

I asked the group to remember a stressful situation and notice what they thought and how they felt. Then we did five minutes of breathing practice. I asked them to notice again how they felt. Most people said they felt much better. One person asked for exercises that energize them during boring meetings.

Breathing techniques calm the flight-or-fight reactions in our mind and body. As the adrenaline stops pumping, we can think more reasonably.

The instructions are easy and flexible:

- Make yourself comfortable.
- Set a timer for at least 5 minutes.
- Breathe in while counting.
- Breathe out while counting.

Count to 1, count to 5 or any number in between. Choose a number that is comfortable but a stretch for you.

When your exhales are longer than your inhale, and you slow down the number of breaths you take per minute, the practice has a calming effect. Faster inhales and exhales, with more rounds per minute, will be energizing.

My favorite variation is box breathing.

- Breathe in for 4.
- Hold for 4.
- Breathe out for 4.
- Continue this for at least 5 minutes.

There are apps that guide you through these exercises. You can add soothing music or white noise in the background. For some people, breath lacks sufficient sensation to keep their attention. You can squeeze a ball, squeeze and release, to help keep your focus.

Whatever method you choose, experiment with breathing before you send an email or take an action. Experiment whenever you are reacting. See what happens.

Go for a Walk

Have you ever worked for six hours trying to find a bug, gave up, went to bed and in the morning, you knew how to fix it?

Given up when stuck, gone out for pizza, then knew exactly what to do when you got back? The same magic works when you are stuck in a reaction. Go for a walk. Stretch, move, dance, ride your bike, play with the dog or cat or hacky sack ... move your mind off the problem and into a physical experience. You'll think better afterwards.

Make a Snack or Take a Nap

There is a handy acronym to consider in moments of reaction: HALT. It means hungry, angry, lonely, or tired. You are more likely to overreact to input from others when any of these qualities are present.

You will struggle to construct a sound response if you are overtired or haven't eaten since yesterday. Take time to eat or cat nap and you'll improve your mental balance. Eat healthy food, take a nap, or connect with a loved one.

If you must act under HALT circumstances, try to do the smallest possible thing. For example, create space by saying "I'll be back in 30 minutes and will respond then."

Write

Set a timer and write out, by hand, everything happening in your mind. Don't edit, just go. This practice is best paired with a physical follow up, like going for a walk. The writing will help you gain insight, the walk will help you relax.

Writing is a method of inquiry. The writing can take any form or no form at all. Letters you never intend to send, modeling or documenting a problem, trying to see it more clearly. The goal isn't to write for sharing, it is to sort out your reactions from your more-trustworthy perspectives.

Notice Your Triggers: They are Clues

A logical fallacy is a flaw in reasoning. For example, "if we adopt cloud technologies, eventually we will get massive bills that will bankrupt us, so we should stick to on-prem hosting." This is called "slippery slope", which equates a single step with inevitable catastrophe. There are many steps in between those two extremes.

The further we move away from linear, incremental reasoning toward system thinking, the more space is created for logical fallacies to flourish. They are a form of reactive reasoning.

Erroneous leaps of logic trigger a spiral of reactive, fallacious thinking. Rather than constructing sound recommendations, when we share them, we trigger a maelstrom of reactions. In the above example, everyone will be distracted by the assertion that cloud adoption leads to bankruptcy, rather than focusing on the problem at hand. Also, when there are gaps in our reasoning, people might lose faith in our thinking, so being knowledge workers, it's very important to notice them.

Advertising and politics depend on the use of logical fallacies; we are exposed to them constantly. They appeal to our baser instincts. We are trained to use logical fallacies rather than to spot them.

We need to spot them. Bugs in our reasoning will inevitably lead to buggy systems design.

In Part 3, we go into specific examples of logical fallacies. For now, notice when your reaction is triggered by an error in logic. No doubt, this is happening more often than you realize.

Summary

Our reactions to other people's thinking disrupt our ability to create inner conceptual integrity and to build sound responses.

When we get swept up in their current, we are, at best, unhelpful. Learning to create space for our reactions, practice with them rather than set situations on fire, makes us better knowledge workers and more-valuable colleagues.

We begin with one goal: notice our reactions. We see that our reactions, usually, don't get what we need. We create space for our thoughts and feelings to exist, without judgment, while we tidy them up and decide what to do.

Sharing opinions is a socially fun form of reacting, sparring with other people's thinking. Being opinion-driven is not systems thinking. It's important to see the limitations of opinions and be discerning about when to share them.

Empathy is a core systems thinking skill. We can be in the same circumstances as other people and experience those circumstances very differently. That's okay, our stories don't have to be the same. We can understand each other without blame. We don't need to overdo our empathy and understanding. We can strike a balance between understanding others and taking important action.

Some practices that help us create space for reacting are:

- Yes, and ... critical listening and cooperation.

- The 24-hour rule
- Breathe
- Go for a walk, eat a snack or take a nap
- Write
- Look for logical fallacies

Practices

For one week, decide to practice “Yes, and” in all situations. Take note of any changes as a result. Also note any time you find it challenging. Why?

The next time you are reacting to something someone says, try one of the practices described here. Notice what happens. If you don’t notice any change, try a different one.

Questions for Your Journal

- Remember the last time you were angry about someone. Set a timer for 10 minutes and write a letter to them (don’t send it)! Do this the next time you are angry with someone. Review the two entries. Are there patterns in your thinking? Do they repeat?
- What helps you when you are off balance, caught up in the drama around you?

- If you could ask for a change in how we provide leadership that will lessen the impact of opinion giving, what would you suggest?

Chapter 6. A System of Learning

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 6th chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at sevans@oreilly.com.

Leaders are designers, stewards, and teachers. They are responsible for building organizations where people continually expand their capabilities to understand complexity, clarify vision, and improve shared mental models - that is, they are responsible for learning.

—Peter Senge

This book is titled *Learning Systems Thinking*. In previous chapters, we talked about systems and we talked about thinking. In this chapter, we will talk about the most important word in the title: learning.

Perhaps you already have images in your mind; classrooms, workbooks, homework, AWS certification exams, passing whiteboard tests. You might imagine following a sourdough bread recipe, taking a painting class, watching a YouTube video on coding Vue components, or joining a woodworking workshop. The feelings you feel when you think about learning will depend on your experiences and predilections. Maybe you enjoy learning; maybe you find it a chore.

Chances are though, whatever you imagine when you think about learning will be somewhat different from what I mean by “learning.” When you finish reading this chapter, I hope that we will have a shared understanding.

In This Chapter

You will learn that systems thinking is inextricably intertwined with learning, it is a core skill. As knowledge workers, we need support for our learning practice so we can develop personal mastery. As systems leaders, we can create structures of

support for learning that nourish ourselves, our teams and the organizations we work within. The concepts we will explore include:

- Learning is inextricably interrelated with systems thinking. Systems thinking is, above all else, an integrated approach to learning.
- Learning is the cultivation of personal mastery.
- Knowledge work is a learning-driven career.
- We need to build structures of support for our learning practices.
- A few core practices will shift our thinking towards lifelong learning and supporting other's learning practices.

The practices you'll discover are:

- Considering why you are a software professional – why is learning important to you?
- Developing personal mastery by prioritizing solitude and approaching your growth as disciplines. Exploring what support you need for learning.
- Practicing metacognition, synthesis, doing deep work, applying what you learn and designing feedback loops.

What is Learning?

The illiterate of the 21st Century will not be those who cannot read and write, but those who cannot learn, unlearn, and relearn.

—Alvin Toffler

Humans are born to learn. Our ancient brains are the same as our modern brains, we learn by conditioning. We react to positive and negative consequences by adapting our behaviors, often unconsciously. We don't touch the stove because the stove is hot and hot hurts. We find food everyday and we avoid threats because we are designed to do so. We learn where to find food and what constitutes a threat.

This type of learning is not the type we rely on most when we develop software systems. We don't follow best practices because we will starve to death or be eaten by a tiger if we don't. (Usually.)



Figure 6-1. From xkcd, a webcomic <https://imgs.xkcd.com/comics/goto.png>

While we learn by conditioning, the type of learning we rely on most as knowledge workers is not innate but fed by experience. We are meaning-making machines. We interact with new knowledge and continuously change the cognitive structures in our mind. If we are self aware and at least somewhat self directed, these interactions are not passive. We think about our thinking, we notice the patterns in our own thinking, a practice also called metacognition. Do I understand this sufficiently? What does this mean? How does this relate to what I already know? What else do I need to know now?

As technologists, we use language, symbols and reasoning to create software. We use spoken languages like English, symbolic languages like numbers and coding languages like Python. Our work is inextricable from language, the structure of communication. We construct concepts (ideas) using symbols and we share them using language. We respond to ideas using more language. We encode our ideas using, well, code.

We learn to write code by incrementally adding symbols, lines of code that generate direct cause and effect. We think, “if this, then what?” We test the code to examine the outcome—does this do what I intended? Does my logic hold up when I run it?

We also learn to code by thinking about the context. Why does this matter? How will people use this? What are their goals? How does this code interact with the software as a whole? How will the infrastructure support (or not) this code? Do I also need to improve the way we are working or the delivery process? Will the code run fast enough? What is “enough”? Can we do other things with this new capability – should it be encapsulated? Etc, etc, etc.

Learning is an iterative process of questioning because we can't be certain of outcomes. While our code generates specific behaviors using predictable logic that physically manifests processes that can be tested — the outcomes can scale to create unpredictable cultural phenomena. For example, we didn't intentionally design doom scrolling, social media indoctrination, annoying public phone conversations or texting during movies ... but here we are. Designing technology doesn't depend on our 50,000 year-old ability to learn where to find food. It depends on our ability to engage with the conceptual world and, whenever possible, predict and *design* patterns of change.

For systems thinkers, learning is holding multiple mental representations simultaneously in your mind, like the code, the software architecture and what the code is meant to do, then

manifesting something from those relationships. Learning is creating new mental representations by integrating your previous experience and knowledge with the novel challenges you are facing. The point of learning is to actually do something with your learning. But keep in mind, there are often multiple ways to both envision a problem and (potentially) solve it. Identifying and exploring those various ways, figuring out what to prioritize, is also learning.

As you can see, I don't view learning as simply following a recipe. Learning is becoming a technology chef.

Learning is Conceptual Architecture

Concepts are the building blocks we use to construct what we build. Imagine that every new idea or insight you experience is a Lego brick. As you learn, you expand the number of information blocks available to you. You increase your ability to build new things with them. You aren't simply adding Information Block A on top of Information Block B. You are forming relationships between them. What is the relationship between A and B? What further information blocks (C,D and E) do you need? For example, you read that low fat diets are the best way to lose weight. You read that high-fat diets are the best

way to lose weight. What do you now know about losing weight?

We can also reshape what is built. You can look at a process constructed by concepts (like a CI/CD pipeline) and reshape it. You can figure out which blocks to remove or what relationships changes will form better patterns. Learning is pattern thinking as well as information gathering.

Software depends on structuring concepts more often than you might realize. A function call, for example, is a conceptual structure used to organize activities for the human mind. We use functions to form conceptual relationships and categorize activities. The code itself, generally speaking, doesn't care.

Our concepts are so interwoven with working software, there is no potential (or reason) to pull them apart. We think therefore we build. We build what we conceive and communicate. Our cognitive processes are the code and vice versa.

Conceptual integrity is the most important consideration in systems design.

—Fred Brooks

Too often, we limit our definition of learning to remembering and applying information. Winning at Jeopardy, for example, or passing the ITIL certification exam. Both demonstrate what a person has learned but not how they apply learning. You can build a Lego tower by following instructions but can you build something novel with those blocks? Most importantly to software development, can you build something new in partnership with others?

Nonlinear learning expands our definition of learning activities to include discourse, thinking and communicating with others. Learning is proactively integrating knowledge and experience – your own and other people's – by structuring inquiry. You craft knowledge not by rote memorization but through pattern recognition, synthesis, hands-on experience, asking questions and the real-world testing of your theories. You wonder and explore. Learning is not something we achieve, a destination, it is at the core of how we live our lives as knowledge workers.

Our *raison d'être*.

I confess that I took this for granted, learning is our purpose. But I discovered that not every software professional agrees with me. My path into technology was all about hands-on learning supported by some academic structure. Solving real-world problems with code taught me to solve real-world

problems with code. Solving real-world problems with code *still* teaches me to solve real-world problems with code. This is not the path everyone takes. Some of us learned skills in a Bootcamp or other programming environment and apply them daily to tasks assigned. Is a developer with 20 years experience in a coding language still learning? Is she expected to be continuously learning?

My answer is yes. The world around us is constantly changing. The problems we solve with software morph into other problems. The structures we build with technology, like air traffic control monitoring and information sharing, are in flux as new problems and new technologies emerge. We might disagree about how much learning we need to do as knowledge workers, but as systems thinkers and designers ... we are always learning from the changes in our environments. We are always examining what we know to see if it's still true, if it perhaps was never true. And, on a more practical level, if we code our whole lives, it is unlikely that we will code in the same language using the same tools. Change is inevitable.

Our linear, limited definition of learning has led us down a primrose path, believing that there is a straightforward, appealing, structured and controllable journey we can take towards expertise. To some extent, this is true. We can learn

GoLang through incremental skill building and experience. Is that expertise? Or is expertise the ability to create matterful things that fit the needs of a circumstances using GoLang skills? That type of expertise is messily acquired and arises when we bridge gaps in knowledge. In systems, avoiding the messy work of learning will not get us where we actually need to go.

NOTE

Systems thinking is, above all else, an integrated approach to learning.

The two core skills discussed in chapters 4 and 5, self awareness and emotional intelligence, are disregarded in a rote-style learning approach. But they are critical to systems thinking. We need those skills if we want to also develop metacognition, the ability to reflect on how we think and learn. Without metacognition, you can not drive your own learning practice; you can only ride in the backseat. And you will struggle to integrate knowledge in meaningful, matterful ways.

Immaturity is the incapacity to use one's intelligence without the guidance of another.

—Immanuel Kant

An integrated approach to learning also includes abilities like:

Cognitive flexibility

Changing your own mind as you consider new perspectives or reconsider old ones.

Inference

Forming and reforming conceptual relationships between ideas and experiences, whether or not those relationships are explicitly communicated.

A tolerance for cognitive dissonance

You can hold two (or more) contradictory ideas in your head, without rationalizing them away.

Pattern recognition (and inductive reasoning)

Understanding how events over time form patterns that then become the structures that inform choices and behaviors.

Categorization

Grouping things together that share something in common, without creating concrete walls around these categories, which are always in flux.

Sound judgement

Discern which information blocks to focus on and which to set aside.

Metacognition helps us develop sound judgment through experience. An amateur tennis player watches the ball as it comes towards her and tries to move her entire body to meet it. A professional tennis player watches her opponent's swing, anticipates the geometry based on racket angle, moves her body just enough to position her racket behind the ball, watches the ball and discerns when to expend more energy and when to let it go by.

A Learning-Driven Career

We will never transform the prevailing system of management without transforming our prevailing system of education. They are the same system.

—Peter M. Senge, *The Fifth Discipline: The Art & Practice of The Learning Organization*

Moving towards a learning-driven career might sound straightforward. But the unconscious impediments to your learning might loom larger than you imagine. We are born

learning but by the time we are five or six years old, we enter educational structures that shape our natural desire to learn into activities that meet external expectations. Our education systems are the training ground for our organizational systems. Learning shifts from a metacognitive process to an expectation-driven process.

NOTE

Thinking of learning as meeting expectations can be a tough mental habit to break.

You may not have noticed how your learning efforts have become industrialized. We begin as raw materials and end up packaged into one role where we apply knowledge while generally staying in our lane, at our desk, being good. Learning becomes associated with classrooms, completing assignments, passing exams (or not), and following career paths starting at 18 years old. Learning is a natural and consistent activity, like breathing, that can be constricted, marshalled and controlled but not stopped.

The industrialization process has taught us to buy packaged curriculum and supervised learning experiences that feed us the information or skills we need to stay competitive in our role. We learn in order to prove ourselves valuable to a system

interested in its own growth. Most of us have accepted this way of life as normal, our learning has one objective, to train ourselves to be a cog in a machine designed to generate revenue. Our actual interests, our deep curiosity and desire to expand past the boundaries of our assignments, are relegated to after-hours hobbies.

This all sounds like a bad thing, but it's not (necessarily). This is a book about systems thinking and part of its job is to help you decide if you are interested in systems thinking. I can tell you that metacognition and continuous learning is a core systems skill, but I can't tell you whether or not you'll enjoy those activities! Perhaps you prefer classroom learning, that's good too. We can't do difficult things, like cure cancer or prevent a climate crisis, without training. We need to form information-sharing materials, like courses and textbooks, and expert groups to decide what are the foundational skills. I want my heart surgeon to be trained in everything she needs to know before she operates on me. We need structures of other people's knowledge, like medical school, to help us discern what we need to know.

My point is only that, for most of us, it is a mistake to expect the organizations you work in to sufficiently nourish and support your learning. Organizations are the end goal of an educational

process aimed at training workers. Your learning is measured, not by your own capacity to learn, but by the market's demand for certain types of knowledge or problem solving skills you already have developed. I'm advocating for alternatives as well, especially learning from Interesting Problems.

Like many technologists, I followed the breadcrumbs of Interesting Problems along a windy career path. In 2005, while learning PHP, I downloaded an open-source CMS for the first time (Drupal), got curious about it and found a client with a problem I could solve using those skills. I continued learning necessary competencies, both technical and social. How do I manage clients? Where do I go for support when a bug won't fix? How do I set up good deployment processes? How do I push data from my dev instance to production without committing mortal sins?

A few years later, I was engineering lead at one of the top Drupal consultancies, building enterprise sites and teaching core competencies at conferences. This outcome isn't magic though I'm sure luck played a role. When you put learning at the center of your career, you scale.

I hope this chapter inspires you to think more about your role as a leader of your own, and other people's, learning.

Meanwhile, If you want to have a learning-driven career, you will need to develop personal mastery.

Developing Personal Mastery

People with a high level of personal mastery live in a continual learning mode. They never “arrive.” [...] Personal mastery is not something you possess. It is a process. It is a lifelong discipline. People with a high level of personal mastery are acutely aware of their ignorance, their incompetence, their growth areas. And they are deeply self-confident. Paradoxical? Only for those who do not see that “the journey is the reward.”

—Peter Senge

Learning, as I define it here, is the cultivation of personal mastery: constantly learning about yourself and the reality in which you exist. I'm saying “personal” mastery because the goal is to improve *how* you think and learn not, specifically, *what* you think. Mastery is a practice, one we never finish or “get right”, that leads towards deeply understanding an area of interest. It is antithetical to “just learn how to do something then do it again and again forever.” Personal mastery is self leadership: continuously developing understanding and skills.

More importantly, continuously defining why you do it. How is your learning practice aligned with your core values? How are you living your values through your work?

What matters most to you?

This is an important question and one worth pausing for a moment to ask yourself: Why are you a software professional? As a knowledge worker, why do you want to develop personal mastery? How do you express your values through your work?

Perhaps your answer is “I build software because I like paying my mortgage.” That is one of my answers, having experienced life without sufficient income. But there are nearly infinite professional roles that would pay me enough. Why this one? The deeper reasons are the ones that motivate our learning.

Developing personal mastery is a discipline. I don’t mean militaristic discipline like sitting in your chair every day at 3pm, back straight, finishing your assignments. I don’t mean control, like turning down that gooey brownie because you are on a diet. (Though, of course, you do need some of that type of discipline.) The word discipline comes from the Latin *disciplina* which means *to learn* – the process and practice of developing competencies. *Disciplina* is learning, learning is a discipline.

Developing personal mastery through almost any set of disciplines will help you think nonlinearly. Because – when you look at any one subject area deeply – you will see systems.

There is no linear alternative, no single course you can take or graduate degree you can pursue that will reward you with personal mastery. You learn by integrating what you read with the talks you hear, your hands-on experience with what you learn from other people's experiences, your analysis of a situation with other people's views. You rely on sound judgment and pattern recognition to decide what to trust and what to ignore.

Then, you share your ideas and see how that goes.

This is important. Developing personal mastery requires generative work, designing the structure of your learning as well as doing something with that learning. You are mastering the ability to do difficult things with other people ... you learn most from doing difficult things (with other people).

Writing this book required showing up at my desk every day (okay, not *every* day, most days) and writing sufficient words. It required grammar skills, professional experience and subject

matter expertise. All of which I get from traditional learning approaches.

Writing this book also required envisioning the chapters and designing the structure (seemingly out of thin air), deciding what to include and what to leave out, understanding the flow of necessary information and how it might be experienced by someone else. Inventing something new from knowledge. That work helps me to develop personal mastery. I'm not just thinking about systems thinking; I'm also thinking about thinking about systems thinking. As I've described previously, I absolutely needed other people's input and support to achieve this vision. True knowledge work is always an integration of knowledge, a team effort, it's never born whole out of the forehead of a smart person.

Prioritize Solitude

What we choose to focus on and what we choose to ignore—plays in defining the quality of our life.

—Cal Newport, *Deep Work*

Yes, we need other people. We also need solitude. Cal Newport defines solitude as “a subjective state in which you’re isolated

from input from other minds.”¹ The readily-available jolt of dopamine distractions of our modern world can, if we let them, completely eliminate solitude from our days. This makes developing personal mastery impossible. Creating time to engage your own thinking, to consider and shape it, is the foundation of learning.

Virginia Woolf said that for a woman to write fiction, she must have a room of her own. My experience has been that the single most powerful thing you can do is to create space for thinking and learning. You do this by creating the why, what, where, when and how of your practice.

You’ve already asked yourself (earlier in this chapter) why you are developing personal mastery. What you will learn is something you will continuously design. Where is the physical space where you go to think and write and whatever else you practice, When is creating space in your day for your practice.

How—that is the implementation details. You can find lots and lots of support for that aspect. Want to learn to play guitar? The internet can show you how, you’ll find many recommendations for how to practice ... but the internet can’t tell you why or where or when. You need to design those.

Owning your own time, energy and attention can be a radical act. Parents of young children, for example, might find it tough to get 20 minutes to themselves a day. When you gather your attention, you might feel the tug of all that you “should” be doing. I still haven’t figured out how to avoid that tug. Somedays, it pulls me away from my desk. What I have learned is that the more I practice prioritizing solitude, the more likely I am to enjoy its benefits.

Mastering Disciplines

If you can't learn, you can't thrive.

– Cal Newport

As knowledge workers, we are fully invested in mastering disciplines. Personal mastery is one of those disciplines. What are the other disciplines we most need to master as software professionals?

This question is a can of worms. Just typing the words, my mind filled up with arguments I’ve heard over this question. Many of us are in the midst of a culture war about what our roles are and what they should be. This “knowing what you are supposed to know” can feel relentless, once you develop mastery in one

area, the next area of interest, process, framework, programming language or social shift is waiting in line for your attention.

The more you know, the more you become acutely aware of how much you don't know. This might trigger anxiety, make you uncomfortable, but it is a good thing. My favorite measure of learning is: are you more aware today of how much you don't know than you were yesterday?

As this is a book about systems thinking specifically, I can deftly sidestep outlining specific skills we need as technology professionals, leaving it for the many books and conference talks grappling with it. Any list I make would be outdated quickly. There are, though, two things I want to highlight.

First, you may have noticed that this chapter quotes Peter Senge often. He wrote the book *The 5th Discipline: The Art and Practice of the Learning Organization*. There is a lot of overlap between what I'm writing here and what he envisions. Three of the five disciplines are *personal mastery*, working with our *mental models* and *systems thinking*.

The other two are:

Shared vision

How we work together towards the same “why, what and how” of a business. In software architecture, keeping the “how” connected to the “why” is a critical practice. One that is often neglected. As software professionals, learning to share our vision, not just our code, helps us contribute to systems design.

Team learning

How we create something that no one person can create alone. This should not be confused with group think or alignment. Team learning is the synergy and synthesis that make good teams great and it arises primarily from the way people on the team communicate.

In my work as a systems architect, I could not function without these five disciplines. More importantly, over my career, every engineering team that was enjoyable and successful demonstrated competency in these five disciplines.

If you imagine that systems architecture equals Kubernetes implementation, you might argue that Kubernetes is a discipline. Kubernetes is an implementation tool. When we practice carpentry or woodworking, we likely use a hammer and a saw but Hammer is not our discipline. Systems thinking (and architecture) are tool-agnostic disciplines that are greatly

aided by, but not limited by, the tools we use. The relationship between our intentions and the tools we use is a factor as well.

Which brings me to the second point. If your career is anything like mine, your disciplines are convergent. Communication skills interact with programming skills; technical implementation skills interact with organizational leadership skills; learning from others interacts with deconstructing your own mental models.

From the day I was born until I was 30 years old, my primary focus was theater. It was my college major, where I invested my creative energy – I even taught classes. I never moved to LA, got a part on Broadway, or for that matter, made a living acting. You could say that I failed and in some ways, I did. In other ways, I benefit every day from that discipline.

When I began my science career, I thought I was quitting acting and writing and other creative pursuits. But of course, there is more overlap than difference.

What are the non-tech disciplines that help you in your technology career? Five skills in particular have been instrumental in mine:

I'm comfortable with public speaking.

I can get up in front of a group and impactfully present ideas. While I'm not Ryan Reynolds, I'm not tedious and boring either. I can usually get people on board with a recommendation if I have the chance to present to them. You likely share your ideas, even if it's just with your team. How comfortable are you with public speaking?

I'm very self aware.

People think acting is being fake when it's actually being deeply real. You must be aware of what your body is doing, what you're feeling, and how you are interacting with others. When I'm able to tap into that authenticity, people experience my presence and sincerity. This makes trust building and team forming more effective. Do you feel comfortable being yourself in your professional relationships?

I'm good at improv.

I can stay cognitively flexible, listening to what others are saying and interacting in ways that, I hope, evolves the ideas. Do you think creatively with others and does it energize you?

I think a lot about backstory.

What you see onstage is the tip of the iceberg, actors are conveying the complexity of the situation even though it's not explicit. When a "scene" or circumstance is playing out around me, I consider what is unsaid and unseen, too. This skill is especially valuable as a systems architect. Are you good at understanding the unspoken stresses and relationships that create blockers?

I will risk making a fool of myself.

Like most people, I'm afraid of public speaking and being authentic and asking for help when I don't know something. Those things aren't easy for me, even after years of practice. I do know though, in my bones, that they are worthwhile, in the long run. Even if I totally flop or ask the stupidest question ever asked, I'll grow. Do you sometimes wish you could take more risks?

I hope this example encourages you to expand your list of "skills that benefit my tech career." The boundaries around what matters to learn are fluid, a dotted line. I don't want you to worry too much about what is career-related learning and what isn't. Personal mastery will always, inevitably, reveal who you are as a whole human. I still see, day to day, that as I become a well-developed human, I become a better knowledge worker. You never know for sure what skills will benefit you most.

I know a “mastery” approach can feel overwhelming. We all are operating at capacity. “Seriously, Diana, now you want me to become some ninja learning personal master too?” No. Honestly, you probably don’t need to apply more *effort*. You just need to shift your core mental model to a “*learning is my thing*” mindset as you go through your day. Start there, create some solitude, see what happens.

You don’t need more willpower, you need to support your learning practice with a system of support.

A System of Support

Through learning we re-create ourselves. Through learning we become able to do something we never were able to do. Through learning we reperceive the world and our relationship to it. Through learning we extend our capacity to create, to be part of the generative process of life.

—Peter M. Senge

If learning isn’t (simply) summoning the willpower to sit down at the table every Saturday afternoon and do Udemy courses what practices support our learning?

Sitting at the table on Saturday afternoon doing a Udemy course might play an important role in your learning life. I usually learn by doing. Building things is a great way to learn, one I highly recommend. How do you learn to code? Code! Find a problem that interests you and figure out how to solve it.

Learning by doing, though, leaves gaps in my knowledge. Especially in areas that are not my specialty, like front end development. I learned Vue by building a front end web application but I don't know how to integrate TypeScript or build forms because I didn't need to do those things. A Vue course showed me what I was missing.

A course can also be a good starting point. As I write this chapter, I am working on a project involving Flutter, which I've never touched. I followed their excellent "build your first app" video before attempting to code anything.

Studying for a certification exam is also a good way to fill in gaps, like learning the semantically-meaningless names for AWS tools. Taking an in-person, hands-on course or online workshop, diving into a degree program or a bootcamp – buying a curriculum and using it to structure your learning is a tried and true method.

It is, however, insufficient if you want to also *understand* something in an intuitive way. Like the guy who came to fix my home heating system and, although he didn't yet know what was broken, he makes a good (and as it turned out, accurate) guess. He had the type of understanding that requires experience with systemic problems, a synthesis of knowledge about different components, pattern recognition and an integration of skills.

In software systems, moving towards personal mastery involves questions and concerns like:

- What are the guiding principles, the most important things to understand, about this subject?
- What is the relationship between this subject and others that are relevant?
- Am I engaging in dialogue about this subject with others? Does that dialogue help me to understand and perhaps even improve the guiding principles? Lively discussions at conferences have been my teachers, including a memorable night in Copenhagen where a small group of us debated configuration management into the wee hours.
- Can I use ubiquitous language (words everyone in the room understands) to talk about solving challenges with these skills? For example, can I describe what people will do with

a front end application without using the word “component” or “widget” (even though that’s what I am probably imagining.)

- Who can help me learn this? Who is thinking deeply about this subject?
- Can (and do) I help others by applying this skill? How? Can (and do) I help others develop this skill?
- Does personal mastery in this subject improve my comfort with complexity? Does it improve my conceptual flexibility? Does it put tools in my toolbox that help me do hard things in changing situations?

Here is the most important question.: Am I developing personal mastery in this subject?

NOTE

As a chef, you might learn how to flambé, but your goal is not to become a flambé developer, hired to flambé, after demonstrating your flambé skills in an interview.. Your goal is to do appetizing things with food, sometimes by setting it on fire.

Learning is inquiry that happens in a nonlinear way. To make space for inquiry at the center of our lives, we need a system of support for our endeavors. What supports my lifelong learning practice is probably different from what will support yours.

There are as many combinations of supportive practices as there are people. Here are some that might help. Mix and match as desired.

You do not rise to the level of your goals. You fall to the level of your systems.

Your goal is your desired outcome. Your system is the collection of daily habits that will get you there.

—James Clear, *Atomic Habits*

Practice Metacognition

Learn how you learn. Notice how you think and react, what improves your thinking (especially your nonlinear thinking) and what holds you back. Discover what time of day is best for learning. I'm a 4am person, as was an engineering colleague. Except that he was still awake at 4am and I was just getting up. You are the #1 expert on you. How do you remember, apply, teach, refine? Your learning practice is, as a knowledge worker, your most valuable asset. Respect it.

A subset of metacognition is *trust yourself even when you are an outlier*. At university, I took a required computer science class that I did not enjoy. There was a final exam that involved

writing code that fought with other people's code and the last man standing got an A. I'm not interested in that activity – it is the antithesis of why I build technology. Everyone seemed to enjoy it except me and the only other woman in the class. In 18+ years, I have never used the skills I learned in that course. Which doesn't mean those skills weren't important! They just weren't important for me. It's okay to need different learning experiences and outcomes than the mainstream. And it's okay to have negative experiences – the stuff you hate teaches what you like.

Proactively Integrate Other People's Thinking With Your Own

Your knowledge is the main branch in a repo and every learning experience is a pull request. You get to decide what and how to merge other people's knowledge with your own. You will learn more from that process than by passively consuming information.

A sub practice is to *proactively fill in gaps*. When you learn something new, seek out more and varied ideas to help you develop understanding. For example, whenever I watch a movie or documentary based on a true story, I research

afterwards. What was accurate and what wasn't? What else was happening at that time (to help me understand context)? When you discover a new interesting topic, find the thinkers that describe the core principles and read or watch their stuff. What is most important to know? What are the pain points?

Apply What You Learn to Real World Challenges

A family member shared an assignment they'd been given in a technology course on front-end development. What struck me was how far away from daily life as a front-end developer the learning activity was. There is still, sadly, a big gap between academic learning and commercial work, at least in my part of the industry. Academic learning can teach us skills that lay a strong foundation for a career. Like learning classical piano even though you want to be in a rock band. Learning on the job can, as we discussed earlier in this chapter, leave gaps. In an ideal world, you will integrate both study and experience.

An extension of this practice is to not overfocus on greenfield (brand new, the world is your oyster) projects and activities. The majority of our work as software professionals is developing new behaviors in legacy code. Learning how to

apply your learning to change or adapt existing software is valuable experience.

Do Deep Work

Concentrate. Focus. This is the meditation practice of knowledge workers. We experience so many demands on our attention, you might not remember what it's like to spend an hour immersed in doing something challenging. If you are always interruptable, you are corrupting your core value as a knowledge worker. You are also training your brains to seek more distraction.

Practice learning without distraction, as long and as often as you can. Even if it's only 20 minutes a day, that's an excellent start.

If you grew up with the internet, you might not have experienced deep focus. In the US, a major movie theater chain is considering allowing texting during movies. The CEO said, "You can't tell a 22-year-old to turn off their cellphone. That's not how they live their life." The problem ignored in that statement is – when you are trying to concentrate, on a movie like Oppenheimer for example, you are distracted by other people's distraction. Concentration can be contagious, as can

the lack of it. If we have something we want to concentrate on, we need an environment that supports concentration.

The specifics of that environment vary from person to person. Creating a space for deep work helps, in your home, in the local library or coworking space, wherever works for you. Marking off time on your calendar, letting people know you aren't available. Turning off your phone or, better yet, you probably need to put it out of the room.

Think of your deep work time as your meditation time.

I use music (without lyrics) to signal "time to focus". As soon as I put on my headphones, my bodymind knows it's time to concentrate. Does that mean I immediately dive in? Not always, somedays I slide to Amazon, real quick, looking for something I suddenly realize I need. Or I check social media and 40 minutes goes by. But the more I train my focus, the more often I catch myself doing this and can bring myself back to the work.

Inviting others to share focus time can also support you. I use Zoom calls where everyone shows up, says hello, then mutes and gets to work. Being in that group helps me stay for the whole time, rather than quit early when I get distracted. Quiet rooms set aside in noisy office spaces encourage deep work.

Paradoxically, background coffee shop noise helps some of us focus.

Explore what works for you but don't give up the first time you feel uncomfortable. Set a timer and stay, even if it's only 20 minutes. Next time, set it for 21 minutes.

Design Feedback Loops

When you are learning something new, it's helpful to get feedback on your progress. A training partner Andrew Harmel-Law, who was also writing a book, read these chapters before I shared them more widely. He pointed out sticky spots, asked questions, made recommendations. His feedback made the writing stronger. A coding partner, Stepan Protsak, never fails to make my code better with his feedback or to help me get out of a stuck spot. Learning partners are our greatest support.

Sadly, the majority of people we interact with can harm, rather than help, our learning process. We share our nascent attempts and get aggressive or demeaning criticism, unhelpful opinions about what we should say or do instead, impatient judgment of our learning process or, and this has happened to me more than once - someone is just plain mean.

We can't avoid the negative attention sharing our work will inevitably attract. You aren't contributing much value if you are only sharing things that everyone else likes and agrees with! But unhelpful feedback should not be integrated too early into our learning process. We need people who can help us do hard things, not knock us down. We want critical feedback but only when it helps us develop personal mastery, not personal shame.

When you ask for feedback, be specific. "Can I tighten up this code?" "Reading this, did you trip over anything or feel confused anywhere?" "What about this is really strong?" "What about this is not quite strong enough yet?" If you don't ask for specific feedback, you are likely to get "this is good!" That won't help you make it gooder.

One Day at a Time, Forever

In higher education, there's a strong bias towards people under 35. (This bias also exists in many tech cultures.) We think of learning as something people do in their 20s. Then they apply those skills for 40 years and retire. A linear line from kindergarten to daily golfing.

Learning, like breathing, is something we continue doing, in various forms, regardless of age or circumstance. If you carry

this bias, I suggest you drop it. When you are 50, 35 will seem still young. When you are 70, 50 is still young. Learning later in life is, in some ways, even more rewarding. I've pivoted three times in my adult life and I'm pivoting again. Who knows if this is the last time.

Learn before you have kids, learn while they grow, learn after they move out. Learn things that you aren't supposed to learn because of your gender or circumstance. Ask the hard questions. Do the thinking you think you can't do. Don't put any restrictions on how much you can learn or what you can learn to do. Enjoy the journey, all of it, while it lasts.

Learning never ends until you do.

Summary

We are born to learn and communicate our learning through symbols and reasoning. As knowledge workers, learning is our core skill. A learning-driven career depends on not simply taking a Udemy course but also developing metacognition – thinking about how you learn.

Systems thinking is inextricable from holding multiple concepts in your mind and forming new relationships between them.

This is learning. We also rely on discourse, the integration of knowledge with experience, and structuring inquiry to improve our conceptual architectures. This approach to learning is personal mastery, expanding not just what we think but how well we think.

To develop personal mastery, we need solitude and focus. We need a system of support around us that encourages metacognition, integrating other people's thinking with our own, designing feedback loops and applying what we learn to real world challenges. We proactively build that system.

If we are lucky, we learn every day of our lives. Enjoying this practice is the key to seeing, knowing and understanding systems.

Practices

As this is a chapter on learning, reading is a great place to begin:

- *Deep Work: Rules for Focused Success in a Distracted World*
by Cal Newport
- *The Fifth Discipline: The Art & Practice of the Learning Organization* by Peter Senge

To help you develop a learning-driven strategy, consider taking the [Top Performer](#) course offered by Scott Young & Cal Newport. Or read Scott's book *Ultralearning: Master Hard Skills, Outsmart the Competition, and Accelerate Your Career*.

Schedule time for solitude. Even 30 minutes, once a day, can begin to open space for thinking and learning.

Create a list of learning resources related to systems thinking and complexity that you might explore. You might begin with *Thinking in Systems* (if you haven't read it yet) by Donella Meadows or *Chaos: Making a New Science* by James Gleick. If there is another discipline you are drawn to, create a list for that instead. Use your solitude time to begin reading, watching, listening or creating something that helps you develop personal mastery in that discipline.

Questions for Your Journal

- Why do you do knowledge work? What is the deepest value you express through your work?
- Imagine an outcome you would love to achieve. What are the competencies someone would need in order to do that work? Investigate options for learning and practicing those

competencies. How will you know when you're nearing personal mastery? What will change?

- What do you think are the five most-valuable disciplines in your area of expertise? Why do you value them? What are the core principles and practices that define those disciplines?

• <https://calnewport.com/spend-more-time-alone/>

Part III. We Are a System of Thinking

Chapter 7. Collective Systemic Reasoning

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 7th chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at sevans@oreilly.com.

“Ambiguity ... presents a puzzle. It challenges one to grasp something from multiple perspectives, each of which demands equal attention. One never solves an ambiguous situation, but puzzles through it, searching for the unity of multiple perspectives, for the sense of the thing this, which is illusive and shifting..“

—Ann M. Pendleton-Jullian, *Design Unbound: Designing for Emergence in a White Water World*

In Chapter 5, I encouraged you to notice your reactions and stop allowing them to drive your thinking. I promised that, later, we would learn how to respond instead. Here we are, in this chapter, we'll build a response.

There are many ways to approach systemic reasoning – we will frame the practice as “creating a recommendation”. The process is deceptively simple but diabolically difficult.

As knowledge workers, we are recommending ideas, actions or theories, formally or informally, all the time. We do this by making artifacts. Artifacts can be documents, models, working code, an email or Slack post, a meeting discussion, sky writing

... the form doesn't matter. What matters is the quality of the thinking it conveys.

Your recommendation will include:

- The idea, action or theory you are recommending.
- Three to five reliable, relevant, sound and cogent reasons that justify your conclusion.
- A clear description of why this idea is highly-impactful and matters right now.

Easy, yes? Try it! Pick an idea and try to create a recommendation. You'll likely discover that the practice is anything but easy, even for expert practitioners. Remember, we are (generally speaking) terrible at nonlinear thinking. Our ideas aren't as well integrated and sound as they feel. Your early drafts will reflect this.

The ideas in your mind seem quite strong ... until you write them down along with the reasons that convinced you. Coming up with strong reasons is always more work than I anticipate. Describing how I “know” something can be tricky. I don’t always know how I know. Sometimes I haven’t developed the vocabulary and conceptual understanding to describe your thinking. I just kinda know, or I think I do, until I examine my

knowing more closely. Sometimes I have an experience that taught me a lesson and that lesson applies.

Other times, I am applying a lesson that doesn't apply in the new context. Or I discover I am wrong or misguided. The inevitable fallacies and weak links in your thinking are interwoven with your quality thinking, like tangles in hair.

You are detangling them.

NOTE

As a practice and a process done with other people, collective systemic reasoning is a driver of systems thinking.

In This Chapter

In this chapter, you will learn how to craft a strong recommendation (idea, action or theory) using systemic reasoning. You will strengthen the reasons supporting your recommendation until you reach the best possible conclusion, under the circumstances. Systemic reasoning is a method of inquiry that encourages you to synthesize knowledge, experience and sound judgment. It also encourages you to structure ambiguity – frame the relevant questions. When you

share your recommendation, others can help you to strengthen the reasons (rather than give you their opinions.) Systemic leadership is developing these skills and using them daily..

The concepts we will explore include:

- What is systemic reasoning and how are we applying it to systems thinking?
- How do you construct a recommendation and strengthen the reasons that support it?
- Using systemic reasoning to learn what you need to know and frame the questions you need to explore.
- Tools like top-down elaboration and that iceberg model that support systemic reasoning.
- Common logical fallacies.

The practices you'll discover are:

- Building a recommendation.
- Strengthening the reasons.
- Using systemic reasoning as a method of inquiry.
- Framing ambiguity.

What is Systemic Reasoning?

The aim of argument, or of discussion, should not be victory, but progress.

—Joseph Joubert, French Moralist and Essayist

Systemic reasoning is the art and science of reasoning systematically in support of an idea, action or theory. When you practice systemic reasoning, you move beyond sharing your opinion. You construct ideas by integrating the reasons that convinced you that your idea is sound, relevant and matterful. You synthesize your own thinking with the thinking of others, arriving at the best possible conclusion, under the circumstances, when conditions are uncertain.

In systems, conditions are always uncertain.

We can rarely know, for sure, what is “right” to do. As knowledge workers, we are usually making an educated guess. Systemic reasoning increases the likelihood that we’ll have a positive impact by cultivating a more cohesiveness, trustworthy and holistic quality of thinking than an opinion. And increases the likelihood that when we take action, we will learn from the results, iterate and improve.

Systemic reasoning is:

A tool, a practice and a mindshift

A tool for constructing ideas, a practice of reasoning skills and a mindshift towards looking at change more holistically.

Reason giving

The core of systemic reasoning is reason giving — supporting and justifying our ideas with reasons that others can evaluate and help to strengthen.

Strengthening the reasons

Weaving ideas together until they have conceptual integrity.

Proactive

Synthesizing diverse knowledge and experience, using sound judgment, into ideas based on valid reasons.

A method of inquiry

Exploring, analyzing, discerning, deliberating, testing and expanding our knowledge.

Difficult to do well

Practicing is essential.

Technology systems are a tapestry of ideas, actions, and theories (concepts) that we have woven together. Systemic reasoning strengthens the bonds (integrity) between the ideas that support our actions or theories. When we strengthen conceptual bonds, we are more likely to make the system itself stronger.

Imagine a knitted sweater that keeps you snuggly warm on a chilly day. A well-reasoned recommendation has similar strength and integrity. Now imagine a sloppily-knitted scarf that unravels at the slightest tug. When the ideas pushed to production lack conceptual integrity, they make a system rickety and frail..

Systemic reasoning enables us to construct ideas, actions or theories supported by as much conceptual integrity as we can muster, despite change and uncertainty. It takes us beyond our usual understanding of problems (and potential solutions), aiming towards insight – seeing what is true about an event in a system. As software professionals, when we look at a problem in a linear (mechanistic) way, we ask:

- What just happened? (We react to the “top of the iceberg” event.)
- Does it work as expected or is it broken?

- How do I fix it or make it do something else?
- How long will it take to fix?

When we look at a problem systematically (nonlinearly), we also ask:

- How do we know, first of all, that it's a “problem”? What is the impact, now and in the future?
- A problem for who?
- How does this problem compare to the other challenges we are currently facing? Is it core to the system’s purpose?
- What has been happening over time?
- What are the structures or processes that keep it happening?
- Is there a shift in our mental models required to alleviate this problem, rather than patch it?

What I discovered, as I learned systems thinking, is that whenever five people are discussing a problem, they are often solving different problems, even as they use the same words to describe it. Too often, I presumed that others already see the world from my perspective. But they can't possibly see my perspective unless I make it visible. Systemic reasoning, done together, uncovers disparities and makes use of them to generate a more trustworthy solution. .

How Systemic Reasoning is Collective

In systems, everything is interdependent. No piece of software lives in a vacuum, isolated in spacetime from all other pieces of software. (Even this statement depends on how you define “a piece software” and “isolated”.) We can rarely think about the system without the support of people building other parts of the system. And unless you are coding alone on a desert island, you aren’t making every decision about what gets pushed to production by yourself. Your thinking interrelates with other people’s thinking.

Software lives in a changing world. Users won’t do the same thing, in the same way, forever. We can’t always predict how changes in the ‘real world’ will impact our technology systems. Which is why we need to think together. The better we understand diverse experiences, the more likely we are to see holistic solutions.

When we reason systematically, we integrate and synthesize diverse points of view. The user experience and the tech reality, the business needs and the real-world constraints. The user experience, tech reality, business needs, real-world constraints and systemic patterns. The relationship between, for example,

my view and a product person's view creates a richer understanding than my view alone.

Thinking collectively, acting cooperatively, does not mean kumbaya, everybody agrees and is so happy together. In systemic reasoning, we engage people who disagree with us – explicitly and purposefully. That's an essential part of the practice. We do this so we can grow understanding and increase knowledge.

Systemic reasoning relies on consent. We change and adapt our own thinking as we learn, grow and engage with other people's insight, experience and expertise. We are open to changing our minds as we listen critically and respectfully. We practice discernment and good judgment when we decide what to accept and what to question. We don't "fix" other people's thinking but we do help others to strengthen their systemic reasoning.

The "winner" in this approach is the system itself. The prize is the benefits we share when our systems are vibrant, healthy and purposeful.

To find patterns and identify what is structuring or influencing those patterns, we will inevitably need subject matter expertise

we don't possess. For example, I learn the most about a system by watching people use it and asking them about pain they are experiencing. This is especially effective when I ask the people building the system "what is holding you back"?

Systems thinking engages mental models and in order to create new mental models, I need to understand the current ones. Thinking and modeling with others, diving deeper into *why* we think what we think, is a fast train to mental model land. When discussions only focus on events, you can't get there.

Systemic reasoning alone is not sufficient if you want to truly understand a system. It is one of the tools in your toolbox. But this practice, done together, is a step towards nonlinear thinking and will help you create a shared understanding of your circumstances.

Recommendations are interrelated, forming a matrix that describes our current system's architecture. When we share ideas, we usually do so in a linear way. "This feature will do X and be delivered on Friday." We get feedback in a RACI structure or we gather other people's opinions.

Collective reasoning is a different approach. A recommendation interlinks information, weaving our ideas with ideas already

shared by others or the code itself. We proactively create this interlinking by synthesizing other people's thinking with our own to support the idea, action or theory.

In chapter 10, Leadership is Synthesizing, we will dive deeper into creating relationships between recommendations. In this chapter, we will learn the skill of building one.

Systemic Reasoning has Other Names

Like "systems thinking", systemic reasoning is a practice that is applied differently in different contexts. We are applying it to software systems, so I am highlighting some aspects and setting aside others. In this chapter, we structure a recommendation because, as knowledge workers, we rely on that skill. If we were lawyers, biologists or climate scientists, we would take a slightly different approach.

To some extent, I am reframing systemic reasoning by integrating practices to fit our context. I'm designing a system (of thinking) that fits our context and serves our purpose. In other words, I'm using systems thinking to structure systems thinking so I can do systems thinking. Systems are meta like that.

In school, you might have learned argumentation and recognize its use here. If you've ever written an essay, taking a position on a hot cultural topics like abortion, civil rights, animal welfare, or wearing a mask during a pandemic, you've been introduced to argumentation. Argumentation is also called informal logic, a practice we often do as software designers, whenever there is no clear "right" way to do something.

Argumentation is not arguing. It is not fighting about who is right and who is wrong. Some practices, like debate, frame it as a "for or against" approach but the practice does not need to be binary.

Argumentation is a social process but it is not persuasion, though we hope people will agree with us. It is not negotiation, finding the most palatable option among people who want different things. Argumentation is supporting claims with strong, sound and cohesive reasoning. We are using this approach to strengthen our systems thinking and communication.

Law and science both draw heavily on argumentation skills. In science, we use evidence-based justifications to support our knowledge claims. Lawyers use adversarial argumentation, where the goal is to win. This "win" approach doesn't help us in

systems. The pursuit of truth is better served by cooperative argumentation, or as I call it collective reasoning, where people practice argumentation with, rather than against, others.

In workshops, I called these practices “nonlinear thinking.” Systems thinking is one of the many ways to explore nonlinear thinking. Other similar concepts include:

- Informal logic
- Critical thinking
- Pattern thinking
- Deductive, inductive and abductive reasoning
- Abstract thinking
- New Rhetoric
- Lateral thinking
- Design thinking
- Strategic thinking
- Working with mental models
- Decision making
- Analytical thinking
- Creative problem solving
- Learning-driven teams (like in the book *The Fifth Discipline*)

You can use any nonlinear thinking approach that will generate insight and build sound, well-crafted ideas. They all help to

create a systemic understanding of the software and the systems around you.

Systemic Reasoning is Building an Idea

1. The practice of building a recommendation is deceptively simple:
2. Write your idea, action, recommendation or theory.
3. Write 3-5 reasons that support it. What convinced you?
4. Include a direct connection to the system's purpose (why does this matter now?)
5. Strengthen your reasons and their relationship to each other until they are cogent – clear, logical and convincing.
6. Describe any known negative consequences, valid disagreements or challenges to current thinking (especially in ways that might be unpopular)?

Figure 7-1 shows a simple template you can use to model your recommendation.

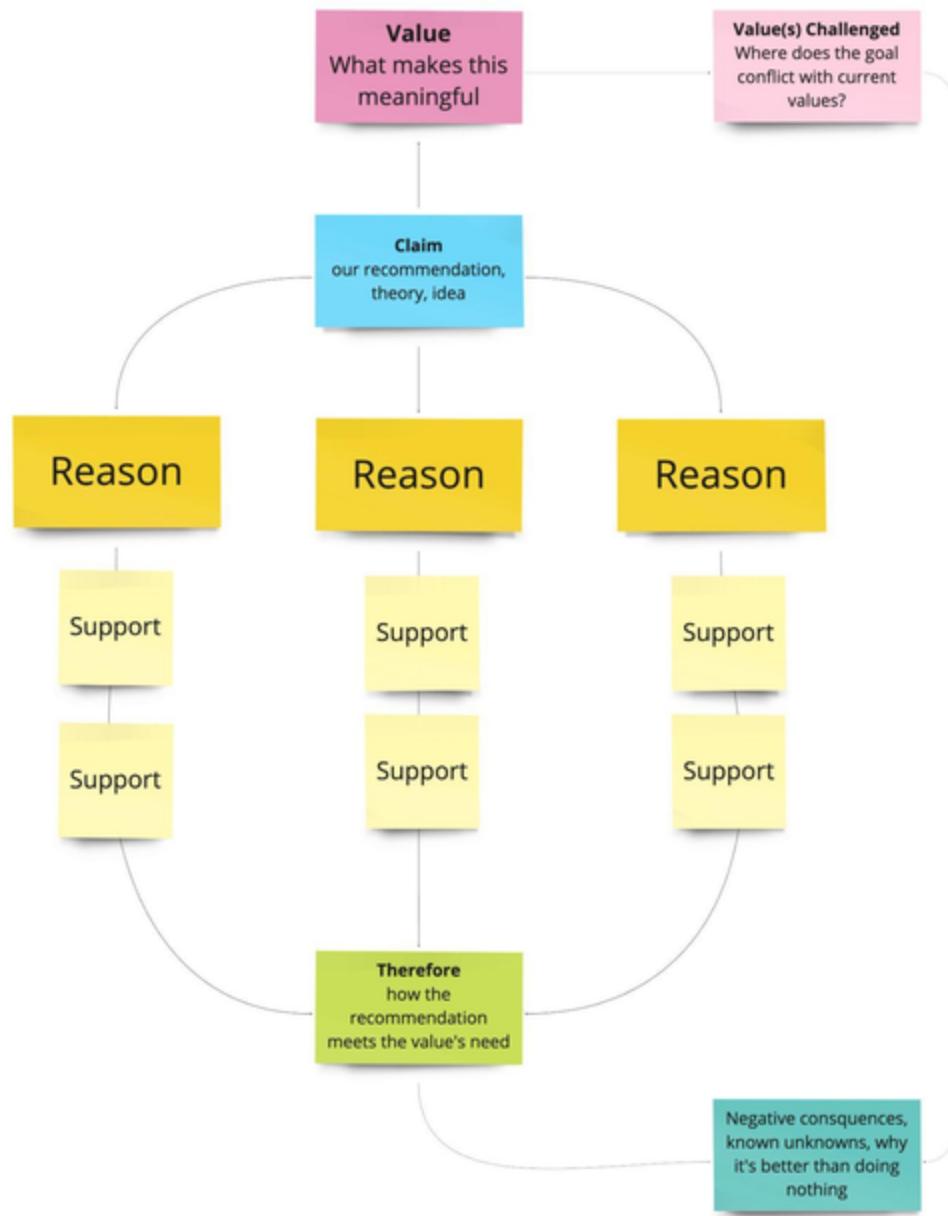


Figure 7-1. is a simple model to structure your thinking

Let's try it! Grab a pen, open a document editor or digital whiteboard ... get ready to capture your thinking and make it

stronger.

Identify Your Idea, Action or Theory

If you woke up tomorrow and knew that everyone would listen to you and trust what you say ... What would you say? What is the idea, action or theory you most want to express?

Your core conclusion, your idea, recommended action, or theory is also called your *claim*. To identify your claim, imagine someone reading your recommendation, nodding and saying “Yes, I agree” ... what are they agreeing with?

Write your claim as clearly as you can.

For example, “I recommend semantically structuring the information we publish to consumers via the API.”

You can improve this claim by including an example. “I recommend semantically structuring the information we publish to consumers via the API, similarly to this example [link].”

When you begin, don’t get too attached to your claim. As you do the work to justify it, your claim will likely change. As you build the relationships between your idea and the reasons that

support it, you will probably edit your claim and your reasons, multiple times.

Right now, your only goal is to clarify to yourself what you think and why you think it.

Identify Your Reasons

“If it is to be a minute speech I shall need four weeks in which to prepare, if a half hour speech, then two weeks, but if I am to talk all day I’m ready now.”

—Rufus Choate, Senator from Massachusetts
USA, 1859

What convinced you that your claim is the best possible conclusion? Using the previous example, what convinced you that semantically structuring information was a good idea and valuable to do now?

Your reasons describe why your claim matters. Why does it matter right now? Why is it a good idea in this context? Your reasons should describe how your recommendation improves the system’s ability to serve its purpose?

If I had a dollar for every time I heard someone say “we should do [something]” or “that won’t work” without any reasoning to justify it, I’d be sitting on a beach reading novels instead of working for a living. If you haven’t seen this pattern at work, you’ve likely seen it on social media – posting as if the speaker having a strong opinion sufficiently justifies their rightness. The problem with these types of assertions is they put everyone else on the defensive. You must change their mind, topple their “rightness” and who wants to do that?

Expanding on our example: “I recommend semantically structuring the information we publish to consumers via the API, similarly to this example [link].” Here are reasons:

Consumers need the information structured to serve their purposes.

They don’t need or want to know how the software structures information to store in the database. They want the information in an understandable-to-them format.

- A few examples: we use four different labels to mean “the title of a piece of information”, because different software teams structured that data differently over time. There are two different date formats. Internally-

understood labels for metadata, like zmby-o, don't describe the information to consumers.

- Adding a story can add emphasis: "Recently, when a customer saw "rtsd-io: Critical" they asked "does this mean a zombie apocalypse is imminent?""

Using a data model is an industry best practice.

We want to follow best practices when they are relevant to our circumstance . Best practices support software parts communicating more effectively and efficiently when separate teams build them.

The CTO recently said that our biggest blocker is “localized, duplicated and hard to get data” in our system. Consistency in our data model and information flow will help resolve this blocker.

Our organization's goal is “to become the goto information source for anyone who benefits from our services”. Reaching that goal depends on our ability to resolve this issue quickly.

There are three practices that help you discover your reasons:

- Bringing together any knowledge or experience that led you to your conclusion.

- Integrating what other people have said that is relevant to your claim.
- Remembering that your goal is to structure a sound recommendation, not convince people through power or overt persuasion.

When building your reasons, let go of the belief that hierarchies raise up the best thinkers. Perhaps this is true, but the CEO can be misguided or incorrect. The intern might see an ideal solution. If you are contradicting what the CEO, or anyone else, has said ... try to first understand the reasons that support their conclusions . Then show how other reasons can support a (different) conclusion.

Too many knowledge workers stay at the surface of ideas. The only way to know a sound (reliable and sensible) and relevant (impactful to the system's goal) idea is to dive deeper into it. As we learned from the iceberg model, systems challenges arise from foundational mental models and systemic patterns keep them in place. If you go through the process of identifying reasons, you are more likely to see some of those patterns and uncover mental models.

Sound ideas are sound because knowledge, experience and valid reasons justify them. Sharing those reasons makes ideas

useful – others can understand *why* a recommendation is (or is not) a sound idea. Then they can help you strengthen your reasons.

Strengthen the Reasons

The most critical step in systemic reasoning is strengthening the reasons that support your claim. You can't be certain — time will tell whether or not your recommendation will have a positive impact. But if you focus on the reasons rather than getting stuck in “right v wrong, yes or no” dualities, you improve the odds that your recommendations will prove valuable.

When you look deeper into the reasons that support your claim, you might discover that you were wrong, or wrongish. That's good! The point isn't only to support your claim ... It's to test it. Talk to others, do some research, consider the relationship between your reasons and your circumstances. Carefully consider opposing points of view, battle test your ideas.

There are many ways to strengthen the reasons, including improving your grammar, adding data and including lively, relevant examples. The four qualities we will focus on are:

- **Reliable:** Are the reasons true?

- **Relevant:** Are the reasons relevant to the circumstances?
- **Cohesive:** Do your reasons work together to form a convincing whole?
- **Cogent:** Can you make your thinking more clear and convincing?

Reliable

Do you need to do some research or hands-on experimentation to ensure that your assertions are true? For example, if you say that “We can use Kafka to solve this problem by doing X” ... ensure that the current version of Kafka does X.

Are other people justified in believing your reason? If you say, for example, that the software stops running for 4.36 seconds at noon EST on Tuesdays when the moon is in Virgo, your fellow technologists aren’t going to take your word for it. You’ll want some hard data to back you up.

Relevant

My favorite example of a not-relevant reason justifying a technology decision is “Netflix does it.” Unless you are a streaming service whose system has evolved in the exact same way to face the exact same challenge ... you’d need to show why this recommendation is relevant to your system and

circumstances. Even a Netflix developer should not simply say “because this is how we do things.” Why do you do things that way? More importantly, why is that relevant *now*?

Context matters. The value of every decision we make depends on the context in which we make it. In *The Lord of the Rings*, Frodo’s journey to destroy the ring is meaningful inside the context of Middle Earth. In a different situation, he’d be a short, hairy guy with apocalyptic hallucinations.

Cohesive

In systems thinking, relationships matter most. Are your reasons well-connected to your recommendation and to each other? When you read them out loud, do they flow, one into the next, leading you to a sound and logical conclusion?

A more subtle way to improve cohesion is to use the same term to describe the same thing throughout your recommendation. And define the critical words you use to ensure that other people will understand exactly what *you* mean. This is advice I struggle to remember. While writing this chapter, for example, my editor pointed out that I interchange the words practice, approach, tools and other words that all mean “this is a thinking in systems activity”. Declutter your thinking by making

a glossary of key words and ensure you've used them consistently.

Cogent

Cogent means that if your reasons are all true – your claim is probably also true. You can test this by reading your reasons and then, “therefore ...” your claim. Does it follow? Given the context and your reasons, does it just make sense to act on your recommendation?

Cogency isn't a hard science, it's more of an art form. It's a blend of reliability, relevance, cohesiveness, connection to context and strong communication skills. The more you practice, the better you'll get at generating cogent recommendations.

A key focus is sufficiency: have you considered your recommendation from multiple points of view? Systemic understanding depends on conveying an integrated view of the situation. Have you looked at this from multiple points of view? Have you made a strong connection between the circumstances and the recommendation? For example, if you are recommending a technology tool (like Kubernetes) have you

described how it will benefit the system as a whole, not simply solve a tech problem?

As you can see, when you strengthen the reasons, you generate essential questions then seek to answer them. The process of figuring out what questions need to be asked is a critical part of systems thinking.

Be Honest About Potential Pitfalls

When you make a recommendation, there are (almost) always other options. Why didn't you choose them? There are (almost) always people who disagree with you. What good reason do they give and why do you think differently?. There are (almost) always potential pitfalls ... things might go horribly wrong.

What are the blockers that threaten to derail your recommendation?

Be honest about all of those things. You don't have to dive in too deeply and create a whole drama but you do need to acknowledge potential pitfalls and unknowns. What are you most unsure about? How will you monitor these risks as you move forward with this idea?

Systemic Reasoning is a Method of Inquiry

The first principle is that you must not fool yourself and you are the easiest person to fool.

—Richard P. Feynman, Physicist

In systems, very few conclusions are Globally and Constantly True. Everything “good” to do depends on the circumstances, the timing, the goals and values of the people involved and the forces acting on the system. “It depends” is the most common answer to any systems question. How do you discover what “it depends” on?

Systemic reasoning is a method of inquiry – a process through which we explore, analyze, discern, deliberate, test and expand our knowledge. Strengthening our reasons involves proactively understanding and gaining knowledge that better supports our recommendations. We are continuously learning and as we explored in Chapter 6, continuous learning is inextricable from systems thinking.

Coming back to our example recommendation: “I recommend semantically structuring the information we publish to consumers via the API, similarly to this example [link].” Here are some potentially-relevant inquiries:

What will consumers do with this information if they get what they want?
What do they want?

What is the current structure of the information?

Do I know what the system's purpose is?
Do the parts of the organization agree?

Do I understand what each piece is (can I describe it at a high-level)?

What is the industry's thinking on structuring data for consumption?

What tools and processes are in use now?

Are there reasons for this structure I can learn from other teams or documentation?

Do these best practices fit our use case?

What tools and processes are available and have we considered them?

What do consumers do with this information?

Is this activity highly valuable?
How is it related to the

What do the people closest to this challenge say about challenges and priorities?

system's
purpose?

Are there considerations that matter in this circumstance, like speed of response or trustworthiness of data?

Who is impacted by this change?

Does this change improve patterns and relationships in the system (as opposed to bandaid a wound)?

When we study a system, we usually learn where leverage points are. But a new system we've never encountered? Well, our counterintuitions aren't that well developed. Give us a few months or years and we'll figure it out.

—Donella Meadows

Inquiry is an investment of time. When faced with complex situations, it takes time to discover an impactful way to intervene. We need to distill the factors involved until they become clear enough to understand. When our recommendations demonstrate an understanding of the factors involved and a reasonable, well-structured conclusion – we increase the likelihood that others will also understand.

A technical problem, like sharing data between software parts, is also a product, business, user, industry, systems and sociotechnical problem. While our role isn't to develop expertise in every area, we can understand the knowledge and experience shared with us by people who play other roles. When we synthesize that information, we craft a recommendation that reflects this integrated understanding.

You might be wondering “Why bother?” That’s a fair question, you are busy enough already. The answer is: you are significantly more likely to convincingly recommend an impactful change. Not because you are politically powerful or charmingly persuasive, but because you truly understand the problem and have arrived (with the help of others) at a cohesive recommendation that is likely to have more impact than applying a patch.

You may feel impatient and want to skip inquiry, favoring a strict focus on working code. You might not enjoy the uncertainty of not knowing; you might even dislike the world outside of your role. Many of us feel caught up in internal civil wars where people who need to think together are, instead, sabotaging and disrespecting each other. Your attempts at inquiry might raise suspicion and “stay in your lane” feedback in organizations that do not foster a learning-driven process.

You might feel that, even as I acknowledge you are busy, I don't really understand how much work you are already doing. Who has time for that?!

When I've asked others to help me with inquiry, I've heard: "We don't do big upfront design." I agree with this sentiment. Systems thinking is antithetical to big upfront. Systems thinking accepts uncertainty rather than attempts to make everything concrete. But if we skip design thinking altogether, we will plant the seeds of problems that will later grow into the issues that are keeping you too busy to think.

It's a vicious cycle.

Weak, unsound or overly simplified solutions are expensive (in the long run). They are tricky to spot because they feel exactly right in the moment. Investing in systemic reasoning saves us significantly more time later. Your iterations will be more "inchng towards a solution" and less "throwing darts at a wall."

The quality of what we build is equal to the quality of our reasoning, think of it like test coverage for ideas. When we leave gaps in our thinking and push them to production, we must later identify those gaps and clean up weeds that have filled them.

Systemic Reasoning Structures (and Frames) Ambiguity

We are constantly structuring ambiguity, making choices and taking action when there is no single, clear option. Whether we are aware of it or not, we couldn't function without this skill. As a knowledge worker especially, our success depends not simply on what we already know but on how we navigate a world full of possibilities.

Here is an example of structuring ambiguity in daily life. Three friends are gathered in a living room on a Saturday night, deciding which movie to watch. They can't be certain which movie they will enjoy because they will watch a movie they've never seen.

They decide on *Bridesmaids* because:

- None of them have seen it,
- Everyone is in the mood for a comedy, and
- *Bridesmaids* has a 90% Rotten Tomatoes rating.

Out of all the possible criteria, they framed the decision with: Let's watch a movie that is new, fun and well reviewed. They could have agreed to watch a movie they knew was good, a

classic film respected by film professors. They could have chosen a story with more-complex human relationships. In that case, Casablanca might have been the right movie for them.

The “rightness” of a decision depends on how you frame it.

NOTE

The focus of systemic reasoning is not perfection – it is acting on reasons that are sound, cohesive and trustworthy.

Perhaps the friends enjoy *Bridesmaids*. Perhaps they are disappointed. Perhaps one friend enjoys it and two are disappointed. Perhaps the power goes out halfway through and they play cards instead. Regardless, they feel the movie was worth a try.

In systems, identifying something “worth a try” is a good outcome. Generally speaking, we feel better about negative outcomes when our reasoning was sound. We know what to try differently next time. The friends might realize they don’t enjoy comedies as much as they imagine they will. Next time, they won’t frame the choice with “comedy” as a structure. They might choose a whole different approach to framing – take turns choosing a movie and watch whatever the night’s

designated person recommends. When we practice the art of framing ambiguity, our conclusions might not be right but they can improve our understanding of what works (or doesn't).

NOTE

In systems, we aren't simply living the experiences, we are simultaneously and explicitly architecting them. Architecture is the art of framing inquiry.

Structuring ambiguity isn't all Saturday night popcorn. When faced with ambiguity, the human mind reacts with fear and anxiety. When we are stuck in reacting, like we explored in Chapter 5, we negotiate with reality, trying to avoid discomfort and ignore what we don't want to face. Our knee-jerk reaction is to create order from perceived chaos by trying to exert control. Structuring and framing ambiguity gives us a less-reactive option and helps us structure a response rather than react.

Many of us learned that software development is making things concrete, nailing things down, moving fast and breaking things. We like knowing, for certain, what to do. We act on the pernicious fallacy that we can somehow outrun uncertainty. Control the future, bend technology systems (and other people) to our will.

Uncertainty is a constant. In systems, we don't try to make it disappear. We act while resting comfortably in the midst of not knowing for sure. And we act again, differently, when we understand things better down the road.

We do this by framing – organizing certain possibilities from the realm of all possibilities. We use systemic reasoning to structure relationships between possible ways of perceiving a situation. We integrate information available to us about how the system operates. We are choosing to tell one story among many.

When done well, we can frame an understanding that magnifies and illuminates the truest, relevant and impactful aspects so we can take reasonable action. There is no right answer but there are answers that are sound.

The Top-Down Elaboration

Here is a simple framing exercise I recommend for nearly all discussions about systemic issues. I've witnessed teams significantly reduce the “noise” that arises during decision making by introducing a single-page top-down elaboration (TDE) as part of their process. Using a TDE, they became more efficient at making decisions and increased the trustworthiness

of their outcomes. More trust meant they gained more control over their day-to-day decisions and more voice in bigger-picture discussions.

A TDE consists of six sections: Summary, Why, What, Who, How and When. Reasonable people disagree on what belongs in each section. It depends! That's okay, this artifact can evolve to fit your circumstances. The more you practice this approach, the quicker you'll adapt and improve it.

Here's how I structure a TDE:

Summary

The organization's stated mission is to share information with anything who will benefit from it. To better serve that mission, we will semantically structure the information we publish to consumers via the API, similarly to this example [link]. While this will impact all customers over time, we will work with a small group of volunteer consumers who will help us iterate and improve. The XYZ team will do the work and coordinate with stakeholders. We will write a service that consumes information from our software's API and structure it to match our emerging data model. We will serve this information via an abstraction layer to our test consumers. The XYZ team has not sized the work

but they estimate it will take them approximately 3 months to deliver the first iteration.

The summary can set the context, if there is a need to give background. It can also summarize the following sections.

When possible, make this a single paragraph. If you are like me, this will be difficult to write succinctly. But writing it will help you distill the essential points.

Why?

The organization's stated mission is to share information with anything who will benefit from it.

Why does this idea, action, theory (recommendation) matter to the system as a whole? Describing WHY an idea is valuable can be difficult. You might not aim high enough and instead focus on a subgoal like saving money. Saving money isn't WHY, unless saving money is the system's, and the organization's, primary purpose.

You might substitute WHAT reasons for your WHY. “A world-class solution for our customers” is a WHAT. Why does “world-class” matter and how would you measure it?

A general rule is: Everyone who reads your WHY will clearly understand the value. Don't make it up! Why does the system exist? What purpose does it serve? In later sections, you will connect the WHAT and HOW to the WHY.

WHY matters. Imagine Frodo throwing The Ring into the river of molten lava. Imagine a woman standing beside a river on a spring day, taking off her wedding ring and throwing it in. Imagine a man in the midnight shadows walking along a river, sirens and police lights in the distance, taking a ring and other trinkets from his pocket and throwing them into the water.

Same WHAT – entirely different WHYS.

What?

To better serve that mission, we will semantically structure the information we publish to consumers via the API, similarly to this example [link].

WHAT —exactly— will we do to meet the WHY goal? This isn't "install Kubernetes". This is "decrease downtime by 20% by rearchitecting the infrastructure to be more resilient."

Who?

While this will impact all customers over time, we will work with a small group of volunteer consumers who will help us iterate and improve. The XYZ team will do the work and coordinate with stakeholders.

WHO is impacted and how do they benefit from the WHAT?

How will you measure it?

WHO is doing the work?

How?

We will write a service that consumes information from our software's API and structure it to match our emerging data model. We will serve this version via an abstraction layer to our test consumers.

HOW will we do, build, design or continue to explore the WHAT? If there are alternative options, describe them. Make a strong conceptual bridge between WHAT into the HOW. You don't have to get into too much detail here, that's for the team to flesh out. Just enough detail to support the idea.

When?

The XYZ team has not sized the work but they estimate it will take them approximately 3 months to deliver the first iteration.

Technologists are often asked, before they have a trustworthy answer, “WHEN will this be delivered?” In reality (especially in systems), knowing WHEN can’t happen before you really dive in. Usually, though, it helps to say something about time.

Everyone, regardless of role, can practice with TDE’s. The next time there is conflict and confusion about what is being delivered, try testing your understanding with a TDE. See if everyone agrees that your TDE describes the work. I have done this with teams that are in conflict and it revealed that they were using the same WHAT to solve different WHYS. I’ve used this with teams that had a tendency to break up into silos. You can write a TDE then put a link to it in the tasks, so whenever questions arise later, everyone can stay on the same page.

Strengthening the Reasons

Miscommunication is the number one cause of all problems; communication is your bridge to other people. Without it, there's nothing.

—Earl Sweatshirt, American Rapper

When you describe the reasons that support your recommendation, chances are, at first, they'll be:

Wrong

Some of your assertions might not be entirely true or accurate. “The most recent update fixed this bug.” (When it didn’t.)

Weakly connected

Your reasons aren’t interrelated. “We need to fix the data structure. It’s a mess. Our users prefer more white space.”

Disconnected from the context

You’ve described what you think but not why it matters. “Kubernetes will make failover faster.” (Without describing why failover needs to be faster.)

Biased

You’ve included ideas you like and none that you don’t like. “Companies with a heart and soul only use open-source tools.”

Opinionated

Your own ideas about how the world should be come through in emotionally-loaded overtones (this is a form of bias). “Agile is a steaming pile of garbage dumped into JIRA.”

Disrespectful or condescending

Anytime you are, directly or indirectly, calling something stupid. “This workflow is a mess because Marketing people got involved.”

Insufficient

The reasoning represents a limited point of view (yours). “Kafka is what we need.” (Without describing its need from a non-tech point of view.)

Vague

The reasons aren’t well connected and don’t follow in a natural order. “We need to make API responses faster. The users are leaving the interface before they finish engaging. The CPO wants more developers focused on javascript.”

Bad first attempts are perfectly normal and likely unavoidable. You feel sure and solid in your thinking until you articulate it and discover ... Your ideas need work. Systems thinking

requires work. I suspect that some technologists reject “abstraction” (condescendingly) not because it’s unnecessary but because it’s difficult and time consuming. Crafting and constructing concepts means climbing over the mountains of your own conditioned ideas and experiences to discover a path beyond them.

Remember from our introduction to systems thinking that systems are counterintuitive. We rarely look at a systemic issue and think “Okay! I see! The challenge is X and we should do Y. Mic drop!” Instead, because of counterintuitiveness, we do exactly the wrong thing and make the situation worse. Our changes have unexpected and unwelcome impacts.

Strengthening our reasoning is a valuable and humbling practice. Fortunately, once you experience how quickly your outcomes improve, you’ll become more patient with the process. You’ll know it really is worth the investment.

When your first attempts are less than stellar, you learn! When you practice systemic reasoning – nothing is lost. Even if you discard your first ten recommendations, or you act on one and the outcome is disappointing, you are honing essential and powerful skills. When you notice gaps, flaws or weaknesses in your reasoning, you are improving your self-awareness, a key

skill discussed in chapter 4. When you don't fully understand something, you will figure out how to figure it out, another key skill discussed in chapter 6. The more you improve your own thinking, the more you improve the sociotechnical systems around by demonstrating conceptual integrity.

You will (probably) do better next time. As long as you stay open to learning – seeing, clarifying, understanding and discovering as you go – you will improve. I've learned more from failure than from success. We are choosing from nearly infinite potential ideas. The process of sorting and prioritizing them is our knowledge work.

NOTE

How do you strengthen your reasons? You make them reliable, sound, relevant, and cohesive. Doing this, alone and with others, improves the reliability, strength, relevance, soundness and cohesiveness of our software systems.

When you strengthen your reasons, you make them more:

- **Understandable:** Well written
- **Reliable:** True and justified
- **Relevant:** Well-connected to each other and the context
- **Cohesive:** Flows as a united whole
- **Cogent:** Convincing

Understandable

Good writing will be more impactful than bad or sloppy writing. The tips that apply to all types of nonfiction writing apply here, so any reading you do about writing well (like *On Writing Well* by Zissner or *Deyer's English* by, well, Dryer) will improve your thinking.

- Avoid jargon. Define any words that don't have a specific or easily-understood meaning.
- Edit edit edit. Partner with someone good at giving editorial feedback.
- Check your grammar.
- Model your recommendation, using the first sentence of each paragraph. Does it still work?

Reliable

Anything that is untrue, or kinda-but-not-really true, will sink your recommendation. True, many of us are living in a “facts don’t matter” information era but if you want to discover systemic improvements, you’ll ensure your facts do.

- Do sufficient research to ensure what you are saying is true.

- Unless everyone involved has the same subject-matter expertise, justify anything you assert. (If you say “graphs don’t scale” you need to justify it.)
- Give relevant examples. Whenever you say that something critical to the recommendation is true, you probably want to give an example.
- Quotes are valuable, use them to support your reasoning.

Relevant

Whenever the connection between your reasons and your recommendation isn’t clear, make it clear. Add descriptions to show that your thinking is relevant to the challenge you are addressing.

- Include the reason that your recommendation has a high-value impact on the system as a whole and matters to pay attention to ... now.
- Describe the situation from a cross-functional point of view. If it’s a tech recommendation, what is the business value?
- Eliminate reasons that don’t have substance. “Nobody likes that idea” is irrelevant unless your recommendation is “only trust ideas that everyone likes.”
- Eliminate generalizations. Phrases like “Everyone knows that ...” are cheats.

Cohesive

Like every system, you are working on the parts but when they come together, the parts should form something larger than their sum. The relationships between your ideas is what makes your recommendation cohesive. It should “hang together” as a whole/

- Does each reason or explanation add to and extend the previous one?
- Are there clear bridges between the recommendation and each of the reasons?
- Get feedback from others who understand the circumstances from a different point of view. Your reasons may be valid but pale in comparison to other forces impacting the situation.
- Look for logical fallacies (see next section) and cognitive biases. They are often hidden in the bridges between your reasons.

Cogent

When all is said and done, is your recommendation convincing? This isn't something you can tweak directly, like fixing a carburetor, but there are some things you can try.

- Put your reasons in a natural order. When you read them, in order, does your conclusion make sense?
- Ask yourself, “If all of my reasons are true, how likely is it that my recommendation is wrong?” Ideally, the answer is “unlikely.”
- Put the idea away for a couple of days. When you come back, read it outloud to yourself. Still work?
- Read it to someone outside of the circumstances. Does it still make sense?

Using the Iceberg Framework

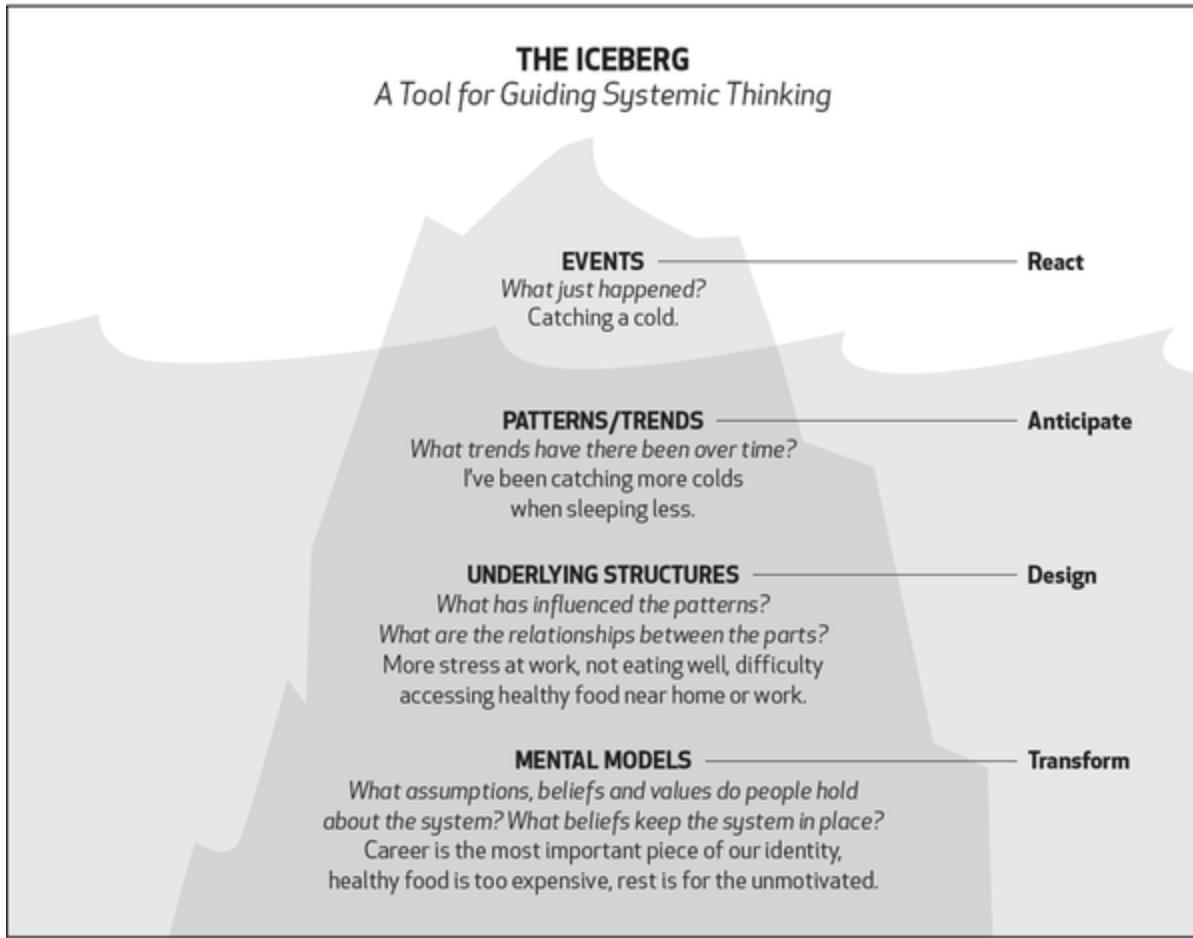


Figure 7-2. Caption to come

Remember the iceberg model from previous chapters? You can use the iceberg framework to help you improve your reasoning. By going deeper into the circumstances, you might see interconnections that you missed. Include them in your reasons.

- How do you know something is happening? What are the signs that an idea, action or theory is needed?

- Is this a unique circumstance or part of a pattern over time?
- If so, what keeps this pattern in place? What are the relationships that encourage or discourage it?
- What are the assumptions, beliefs or values that support or contribute to this pattern? What values or beliefs does your recommendation challenge?

Generally speaking, recommendations that change fundamental beliefs, the core mental models, will have more positive impact than the ones that react to an event. If you can move your recommendation down the iceberg model, you will contribute more value.

Summary

Systemic reasoning is the art and science of reasoning systematically in support of an idea, action or theory. When you practice systemic reasoning, you move beyond sharing your opinion. You construct ideas by integrating the reasons that convinced you that your idea is sound, relevant and matterful. By synthesizing your own thinking with the thinking of others, your goal is to arrive at the best possible conclusion, under the circumstances, when conditions are uncertain.

In systems, conditions are always uncertain.

The practice of building a recommendation is deceptively simple: Write your idea, action, recommendation or theory. Write 3-5 reasons that support it. What convinced you? Strengthen your reasons and their relationship to each other until they are cogent – reliable, relevant, cohesive, and cogent.

When you practice systemic reasoning, you are learning what you need to know through inquiry. You integrate and synthesize diverse points of view. For example, the relationship between a developer's view, a product owner's view and a user's experience generates a richer understanding than the developer's view alone. You are framing the most-relevant questions to answer (aka structuring ambiguity).

The practice is a social process and improves the way we think together. Logical fallacies are the stumbling blocks we need to avoid. You'll discover that they are common, in your own mind and in the thinking around you.

Practices

Build a recommendation. Strengthen the reasons. Inquire. Look for logical fallacies.

Build a recommendation:

- Write your idea, action, recommendation or theory.
- Write 3-5 reasons that support it. What convinced you?
- Include a direct connection to the system's purpose (why does this matter now?)

Strengthen the reasons:

- Edit the writing to make it more clear. Define key words you use.
- Examine your reasons – are they true? Do others know they are true or will they need more data to believe you?
- Improve the connection between your recommendation and the system's purpose. Improve the connection and flow between the reasons.
- Read the recommendation out loud. Does it “hang together” as one convincing unit of thinking? Can you improve the flow?

Inquire:

- What else do you need to understand and how can you find out? Who else can you engage?
- Have you looked at the situation from multiple perspectives?

- What are the potential pitfalls? Have you tried to illuminate them?
- What happens if you ask different questions? Would you come to the same conclusion?

Review the list of logical fallacies. Do you see any of them in your recommendation? Is there any “leap of faith” you are asking people to make? If so, show how you made that leap yourself.

Questions for Your Journal

- If you knew that everyone would listen to you, what are 10 things you would say?
- What are four examples of logical fallacies that I’ve recently experienced? Which ones drive me crazy and how would I rephrase them?
- Why do I feel impatient with taking time to think something through?
- What are the least helpful recommendations I’ve heard recently and why were they weak?
- Who do I know that is good at this skill? What do they do that shows me they are good at it?

Chapter 8. Designing Feedback Loops

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 8th chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at sevans@oreilly.com.

“Our society lacks a feedback loop for controlling technology: a way to gauge intended effects from actual effects later on.

—Kevin Kelly, founding editor of *Wired*

In systems, you rarely arrive at sound conclusions alone. When you take a systems view, there is too much complexity to rely

only on what you already know. You need to learn from other people's experience and expertise.

You need more information about how the system operates. You need to understand the circumstances from multiple points of view. You need the friction that arises when your ideas bump up against other people's ideas, that's where you'll discover outworn mental models. You need the support of collective practices that illuminate, rather than reinforce, your blindspots and biases.

Collective (systemic) reasoning is weaving knowledge and experience, yours and other people's, into strong recommendations and insights. When people work together to strengthen the reasons that support their conclusions, they can think, understand and act in ways that don't rely on persuasion, power or politics.

This is important. The core of self-organizing and empowered teams is the ability to reach sound and trustworthy conclusions, without having to be told what to do. It isn't enough to want intellectual freedom, we need to make power politics unnecessary by building the capacity to create conceptual integrity.

When we create conceptual integrity between people, we push that integrity to production. When we can't think well together (especially when we are divisive), we push changes that, over time, architect a system that is equally weak and unsound. Gaps in our collective reasoning create silos and other unbridgeable canyons in our systems design.

We need to build conceptual bridges over those gaps. Together. In this chapter, we will explore how to design feedback systems and conceptual bridges that improve our ability to think in systems.

In This Chapter

In this chapter, you will learn the value of building conceptual bridges and the common gaps they span. You will cultivate bridge-building habits, the most important one being designing feedback loops for systems thinking. You'll learn the four core skills involved: listening, thinking and considering, engaging with the reasons and avoiding the pitfalls of logical fallacies. Systemic leadership is nourishing and supporting systems-thinking feedback in yourself, your teams and the world around you.

The concepts we will explore include:

- Building conceptual bridges that span gaps in our understanding.
- Why feedback loops—interactions between people about systems thinking—improve our thinking and how to design them.
- Core skills and practices for engaging in feedback loops.
- Common logical fallacies that trip us up when thinking in systems.

The practices you'll discover are:

- Designing feedback loops for your thinking and recommendations: What feedback do you need? Who might give it to you? How will you measure the impact and learn from this decision?
- Discerning what you will take and what you will leave. Deciding when you need to gather more to improve your recommendation.
- The four core skills Listening, thinking and engaging with the reasons when giving feedback to someone else
- Noticing logical fallacies.

Building Conceptual Bridges

As knowledge workers, whenever we are frustrated with our current circumstances, it's important to remember that they arose from our mental models. Our legacy software systems reflect back the values and beliefs that we prioritized. That's not wrong, that's how everything works. To change our circumstances, we can't simply change our toolset, we need to collectively challenge and change those mental models. We need to look into our own minds, at our own beliefs and assumptions. We also need to build new mental models together.

In systems thinking workshops, I introduce people to Mago, the fictitious company I described in Chapter 3. Mago's Quandary is “How do we design a system that meets the demands of the modern systems age?” Their legacy software is becoming dangerously obsolete. They have built newer software on top of the older software in an ever-expanding Rube Goldberg machine¹. Now they want “digital transformation” by adopting cloud tools. But their system represents their mental models.

I ask attendees to dive down into the mental models that might be generating the structures and patterns – the reinforcing feedback loops – that gave rise to current events. What are some beliefs and values that may have given rise to their

current circumstances? Here are some examples of their answers:

- We have to keep up with trends as fast as we can, as cheaply as we can.
- We must react immediately to market changes or “faults” in our system, while still meeting current expectations.
- Don’t touch what is working!
- We are not an IT company. (We don’t need to invest in software development.)
- Our pre-digital processes and organization structure will work well for digital product delivery, we just need to add a few more teams..
- Doing what Netflix does or what McKinsey says or following the hot new trend is a good strategy. (Doing something modern will make our system modern.)
- My group’s goals and incentives matter most. I need to get everyone else to do what I want them to do.
- I fear moving out of my comfort zone. I am uncomfortable moving out of my established role.
- Teams should be able to work in isolation; management should give them work to do.
- Return on investment matters more than quality products.

If we try to change a system without bridging the gaps between our old and new mental models, or resolving divisive core beliefs, changes won't have lasting impact. Sometimes these gaps are purposefully reinforced - leadership might want to keep people apart so they are easier to control. But most of the time, conceptual gaps reflect our inability (or unwillingness) to think well together.

When we can't communicate well, we build software parts that don't communicate well. We constantly try to change the system but we don't change the way we communicate. That is a recipe for disaster.

Thinking with others is like rowing a boat. You won't get anywhere if everyone is rowing in different directions. Collective systemic reasoning enables and empowers everyone to row together without over-reliance on the Captain's Orders to provide direction. Like a restful Saturday paddling on a lake, we can usually figure out where to go. Which is why it's a skill that everyone, regardless of role, needs to cultivate.

Mind the Gaps

Building conceptual bridges between people who are thinking about a systemic challenge is difficult. Really difficult. Even

when we want to do it, we are blocked by some inherent challenges. Here are some examples:

Point of View Blindness

When four people can't agree on a solution they are usually, unbeknownst to them, solving different problems. People with differing points of view conceptualize ecosystems differently. We can experience the same circumstances and use the same words but mean very different things when we use them. When deciding what is valuable to do, for example, accountants and engineers rely on disparate conceptual models when thinking about whether or not to buy cloud technology.

Using a top-down elaboration can help strengthen this bridge. I will describe this straightforward process later in this chapter.

Specialized Language Barriers

A necessity for effective collaboration is a common language. If we use words that have different meanings to different people, or words that others don't understand then we hinder our ability to work as an effective group.

—Simon Wardley

The software world depends on specialized language, often across multiple domains. Product doesn't speak tech, tech doesn't speak business, business doesn't speak systems architecture, architects don't speak accounting. Integrating languages across a domain requires developing a shared conceptual vocabulary. Many times, I've used a word that had a clear meaning for me (like Agile) — only to discover that I was completely misunderstood. The word had a different, antithetical meaning for others.

Some things that help:

- Develop ubiquitous language, as described in Eric Evans' book *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Or in Vladdik Khonov's O'Reilly book: *Learning Domain-Driven Design*.
- Create a glossary of unambiguous vocabulary that is shared by everyone involved. For example, do you use the words user, customer, client, subscriber, and people to describe the same person? This causes unnecessary confusion.

Resistance

Frustratingly, in my experience, resistance is the most pernicious barrier in many tech cultures to thinking well

together. People don't want to interrelate and interdepend. They resist or reject communication work as "soft" unless they are assessing the correctness of someone else's thinking. Some people prefer command and control structures. They prefer isolating their themselves or their team, building a conceptual moat to keep out "non technical" roles or other engineers rather than partner proactively.

I've read books and attended talks that suggest everyone working in tech would be truly delightful to interact with, if only they are understood and empowered. This is, sadly, not the case with any human groups. Sometimes, people are purposefully harmful or difficult. I've seen a flood of derisive condescension blocking collective reasoning, more than once. This can be overt or passive, like meetings where a few participants focus on what is personally relevant and otherwise, don't contribute.

Organizations sometimes reinforce these behaviors, even reward them. In cultures I've experienced that I would call toxic, I rarely saw anyone held accountable for refusal to collaborate or condescending communication. Backchanneling, secret conversations that disagree, disparage or foment resistance, replaced transparent goodwill. Backchanneling

became necessary, anyone trying to get things done had to be subversive, at least sometimes

We can each personally represent and demonstrate the value of interrelation. But we also need to discourage harmful behavior from becoming the norm in teams and organizations. When we encounter “won’t fix” toxicity, more communication doesn’t fix it, accountability does. As knowledge workers, it is fair to expect each one of us to be excellent at sharing and structuring knowledge cross-functionally.

Lack of Structured Transparency

The smallest patterns (which are visible to a team) scale to become the biggest and most impactful patterns (which are visible to the organization). The big and the small are in relationship, shaping each other. The way people interact on teams sets the tone for the whole organization. The whole organization sets the tone for the way people interact on teams.

When closed-door leadership decisions “trickle down” as tasks, true change is unlikely. If “leadership” has come up with a solution but has never interacted with the people who will implement the solution, I’d view that as a red flag and an opportunity for change. When parts of an organization, like

product and tech, are divorced from each other's thinking process, conceptual integrity in the code is impossible. If recommendations can't be modeled by a cross-functional group of people, they can't represent systems thinking.

What an organization knows is not simply the sum of what individuals in an organization know. What an organization knows, and can learn, arises from interactions between people up and down the macro-micro scale. Systems thinking happens between parts of the organization, which means communication needs to happen there. When information is siloed or controlled, the organization limits its own growth.

In order to understand how to do something, we need to understand why it matters to do that thing. We need the concrete and the abstract. Decoupling the “why” from the “how” is a common organizational habit and, from a systems point of view, counterproductive. Success depends on transparency across the sociotechnical system because it depends on a shared definition of success.

We are all building technology systems that act out a story. We are making that story up together, everytime we decide what to do and how to do it. The more we understand the story, and

know how to evolve it, the more impactful our role can be in its unfolding.

That story is playing out on the world stage. We define an organization as a boundary between “inside and outside” but in fact, that boundary is semipermeable. External forces play a key role in the story that evolves inside an organization over time. Nobody is certain, not even the CEO, how organizational decisions will be impacted in the world around it. Things change. When everyone understands the forces acting on organizational goals, everyone is empowered to set better day-to-day priorities. When we have a shared understanding of what we are trying to accomplish together, we can respond more impactfully to new information.

Transparency isn’t simply a culture-building value, it’s a business imperative. People who understand the thinking behind a technology system’s strategic purpose can contribute more valuable work. They can recommend changes to the sociotechnical system that will lead to impactful and meaningful change. Without informational transparency, making recommendations is like throwing darts at a wall blindfolded

The Bridges

In the face of greater complexity today (...) intuition, intellect, and charisma are no longer enough. Leaders need tools and approaches to guide their firms through less familiar waters.”

—David J. Snowden

Command and control approaches, do what we say when we tell you to do it, only work when teams are inherently weak or directionless and need to be commanded. Most teams are, or could be, strong if the communication process supports their strength. Systems leadership encourages systems thinking to flourish with approaches that strengthen everyone’s ability to contribute strong thinking.

Examples of bridge-building habits to cultivate:

- Before diving into solutionizing, **define the problem** and why it’s valuable to solve. When a discussion gets lost in circular email threads, go back to this step. Does everyone agree on the problem and why it’s valuable?
- Avoid the curse of knowledge and **define the words you use**, especially if they seem obvious to you. Even better, use common language to describe domain-specific concepts or geekspeak. People don’t need to know the words ‘Agile’,

‘EC2’, ‘JSON’, or ‘value flow’ to understand what they are and why they matter.

- **Gather others’ expertise** because it is an essential part of decision making². Critically listen to others’ ideas and reflect on how well (or not) they solve the problem. Understand and articulate why they view the circumstance differently.
- Decide that **your goal is not to be right; your goal is to be correct under the circumstances**. We can get overly concerned with getting what we want or proving that our solution was right. We need to balance that impulse with the desire to figure out what is right for the system as a whole.
- **Share thinking and discussions** by making artifacts. A quick document, model, or photo of the whiteboard are deliverables. Make sharing information a habit.
- **Structure a digital space** for those artifacts. Whenever possible, create a transparent view of the system and the strategies evolving within it. Make it easy to connect smaller discussions to the big picture. Evolve a digital space that can interconnect groups of spaces, backlogs or repositories. Use content-sharing tools to make transparency easy.

Feedback Loops are the Net

Often, in systems, there is no right answer. There is no one answer everyone will like. We are making the best possible decision under the circumstances. Which means we need to define the “best possible” and the “circumstances.

In other words, we guess. We make a parachute out of other people’s feedback, gathering information and experiences, and then ... leap. Collective decision making is leaping together.

Everyone involved in a decision is taking the risk. Perhaps they are wrong, perhaps the circumstances will change and it will fail. Perhaps they’ve missed something or there was a better choice they didn’t consider. Regardless, they hope that the choices they make together are better than they’d make alone.

Collective decision making is an exercise in shared vulnerability and critical listening. The goal of working together is to strengthen the reasons. The reasons behind a choice are more important than the choice itself. Do our reasons support our choice? Do they suggest we are heading towards the best possible outcome? The ideal outcome is to co-create reasons that so strongly justify the choice, no further inquiry is necessary.

Are Your Bridges Strong?

How do you know if collective decision making is improving technology initiatives? Here are some indicators that good thinking is taking root:

STRONG	WEAK
Proactive culture and communication	Reactive culture and communication
Different parts of the ecosystem have a similar feel	Different parts of the ecosystem are unnecessarily dissimilar
Solutions are argued at the conceptual level, “Portability is valuable and a priority because ...”	Solutions are argued at the implementation level, “We need Kubernetes.”
Constraints strengthen	Constraints control
Tradeoffs are made	Blame is shifted
Mission driven	Power driven
Transparency	Need to know
Information is shared	Information is a power currency
Decisions can be traced to priorities	Decisions can be traced to authority

STRONG	WEAK
Considerate and respectful	Derisive and frustrating
Elegant simplicity in the system	Mounting technical debt and workarounds

Conceptual bridges bear the weight of feedback loops and help make them effective and efficient. We've explored why we need these bridges ... Now let's talk about feedback loops.

Systems Thinking Needs Feedback Loops

In Chapter 9, we will focus on patterns. Inherent in patterns are feedback loops. When you hear “feedback loops” you might be thinking about, for example, monitoring. Notifications that tell you when a part of your software system is under duress. You might think of autoscaling. These are visible feedback loops that you can design. When you measure user satisfaction, you are creating a feedback loop. We will talk more about those types of feedback loops in the next chapter, when we talk about patterns.

We are concerned, in this part of the book, with designing a system of thinking, So I am focusing here on thinking feedback loops. Healthy thinking feedback loops:

- Gives you the feedback you need to improve systemic reasoning.
- Provides useful feedback (derision is never useful.)
- Proactively engages people with relevant expertise, insight or experience.
- Always involves consent ... is not a way to tell people what they should think but instead, shows them thinking supported by sound reasoning and sufficient evidence.
- Includes a way to measure your recommendation and learn from it.

How will you know that a systemic change accomplished what was intended? How will you discern whether or not your reasoning was accurate? How will you learn what thinking to change?

When I talk about getting feedback, I don't mean the linear processes that usually come to mind. I don't mean (necessarily), the RACI model, a hierarchical structure for getting buy-in, including people deemed responsible, accountable, and consulted along with people who need to be informed. While

this process might, in some organizations, also generate good feedback loops, by itself it is simply a management sign off process. The RACI model presumes that people on a hierarchical ladder are inherently good at giving valuable feedback. This has not always been my experience.

I don't (necessarily) mean an RFC, request for comment, process. Comments are, in my experience, most likely to be opinions rather than true engagement in systemic reasoning. If you are taking an opinion poll, the RFC process works. But for deeper work, without some clear guidance on giving meaningful feedback, seeking comments is not systemic reasoning, it is a survey (which might be a helpful step!)

I don't (necessarily) mean asking your manager how you can improve. Managers are too often taught that "giving feedback" means criticizing performance or improving compliance. While there are circumstances when this type of feedback might be helpful, when it comes to systemic reasoning, it can be presumptuous. Does the manager always think better about the system than people on a lower tier? This has also not been my experience.

You need feedback that makes your recommendation, your systems thinking, stronger. You want to strengthen the reasons

that support (or strengthen) the recommendation. You want to know if you are about to hit an iceberg, if the reasons are truthful, if you forgot something, if other priorities compete etc. Which means you'll need to design feedback loops, depending on what you are thinking about.

As we discussed in Chapter 7 your goal is to make the recommendation more understandable, reliable, relevant, cohesive and cogent. Pause for a moment and remember the last time you got feedback ... Did it accomplish this goal?

How Do You Design One?

Chances are, the feedback loops you need don't already exist. Perhaps they do exist but can be tweaked to include more systems thinking. Whatever your circumstances, your feedback experience will improve if *what you ask for* improves. You don't (simply) want other people's opinions. You don't (simply) want agreement and approval (at this point in the process anyway). You don't want admiration and validation, lovely as those things are, especially when you have a genuinely good idea and a strong, matterful recommendation.

You want to strengthen the reasons.

Ask for the Feedback You Need

There are a few people in my life who are great at giving feedback on my thinking. They rarely fail to suggest some improvement. With those people, I can simply share a link. “What do you think?” They know what to do.

With everyone else, or with those trusted people when I’m trying something novel, I need to ask for what I want to get back. If you change nothing else in your current situation: Change this. Ask questions that demonstrate the type of response you want.

When you share your recommendation with someone, don’t simply ask for feedback. Ask specific questions. Here are some questions you can ask the person giving feedback:

- Have I sufficiently understood the circumstances? What am I missing?
- Is what I’m saying factually correct?
- Did anything confuse you? Did you trip on anything?
- What is the point, as you understood it? (Asking someone to repeat your conclusions back to you is an enlightening and humbling experience. This feedback loop will help you shape your thinking by revealing conceptual gaps.)
- Does the conclusion make sense to you? (Notice that this is different from “do you agree?” Your higher-value goal is to

ensure the thinking is coherent before you begin to unpack it with others.)

Once you craft the feedback you need, you can identify the people who might give it to you.

Get Feedback from People You Need

Systems thinking is a communication skill that, as we explored in Part 1, is not commonly practiced. Nearly everyone can benefit from improved systems-thinking feedback skills. And nearly everyone, regardless of their role, is sometimes biased or blind. Designing a feedback group with divergent points of view is a better way to strengthen your reasoning than (only) climbing a linear ladder.

Who do you ask for feedback? Imagine that you have written down your idea, action or theory (recommendation) and now you want to strengthen it. You might consider asking the following types of people:

- When you are communicating in writing, find people who can edit the writing itself. You might be surprised by how impactful a little copyediting can be when you don't feel understood. If there are models, ask for design feedback.

- People who are subject matter experts. Even if you are certain that you've thought of everything, getting feedback is like getting a good code review. Someone has your back and looks for gaps you might not have seen.
- People who experience the situation from other points of view. As software professionals, we look at most challenges from a limited point of view, one that over-emphasizes the technology challenges. Getting feedback from people who are impacted, but not coding the solution, generates a more holistic understanding.
- People who are impacted. If you are recommending a systemic change that will impact how other teams work, ask them to describe that impact rather than presume you know.
- People who are your intended audience. If you will, at some point, need people to agree with your assessment, ask them up front for feedback. Ask them “how can I make this stronger?” rather than “are you ready to agree?” You will find the path towards agreement, down the road, smoother if you get early feedback.
- People who will pay for it. Very few things I recommend are free ... they will require a budget discussion. Getting early feedback from people who will get me the money helps to structure the reasons to include financial viability.

You can make a simple model. Name five people you will ask for feedback. List two or three specific questions you will ask them. Notice any relevant patterns in their responses.

Follow the Golden Rule of Feedback

As an actor, writer, speaker and software engineer, I've received a lot of feedback. Some of it has been instrumental in developing my ability to deliver good work. A lot of it has been ... let's go with "unhelpful." Not everyone has the skills, temperament, insight or relevant experience to give you good feedback. An excellent coder, for example, might be a terrible code reviewer. That's okay, our goal is not to judge other people's feedback skills (necessarily). Our goal is to discern which feedback to prioritize.

NOTE

The Golden Rule of feedback is *take what you need and leave the rest.*

Whenever I give a talk at a tech conference, the organizers ask for feedback. This can (and has) helped me give better talks. Inevitably though, someone leaves a "this sucked, I didn't like it" note with no reasons. Perhaps it was not their cup of tea. Perhaps they prefer talks with only code samples. Perhaps they

didn't like my style or personality or ... I don't know. I don't know because the feedback didn't tell me. This doesn't do me (or the feedback giver) any good.

Conversely, I often get "I loved this talk!" responses. While I enjoy those more, they also don't help me. What did they love? How did it help them work better? The feedback I need, the feedback I take, is the feedback that helps me give a better talk next time.

Notice that I'm not saying "take what you want and leave the rest." Some of the most difficult feedback can sometimes be the most enlightening. People have given me feedback on talks that "murdered my darlings", a phrase that means "delete something you love.". A joke I laughed at that isn't funny, a point that was too vague, a slide that didn't convey my meaning. I've written many clever, oh so clever, lines of code that didn't make it through code review because they weren't as clever as they seemed.

Sometimes you won't simply murder your darlings, you'll change direction entirely. I have always loved to sing. I have sung in choruses and bands, in musicals, morning showers and on road trips for decades. Many times, I've imagined myself on a big stage signing my heart out. Alas, I am not a very good

singer. I was good enough as an amateur but once I started competing for opportunities with professionals, I saw my weaknesses. Fortunately, I received honest and constructive feedback about how to improve my skills.

This trustworthy feedback eventually helped me decide not to continue trying to improve. Not every goal we want to pursue is viable, not every idea is a good one. Negative feedback is as valuable as positive feedback if the feedback is helpful. “You suck” isn’t helpful. “Let’s decrease the vibrato in your voice when singing that song” is helpful.

When we seek feedback – we *want* to know what doesn’t work as well as what does. This doesn’t mean, however, that we must believe (or even deeply consider) that we can’t sing or our talk sucks or our idea won’t work or we are wrong about a systems challenge just because someone tells us so. Inherent in the feedback process is consent. We get to consider the input and decide what to accept and what to appreciatively ignore.

When you have a cold, people might tell you to take lots of Vitamin C. This advice was made popular by Linus Pauling in the 1970s. Vitamin C won’t have any effect on your cold. This erroneous idea was believed by experts for decades. I believed it and took Vitamin C when I was sick. Ideas are organic like

that, we prune and transform them, even the well-established ones, as we learn and grow. We are all part of a system of ideas that are evolving. I wasn't just told to take Vitamin C, I consented. And now that I know better, I wouldn't consent. I would recognize that the person giving the advice was incorrect.

WARNING

Caution: I don't mean disregarding expert advice (during, for example, a pandemic) or valid science because you've decided, without reason, to disregard it. You can't have expertise in everything, to some extent we need to trust each other's expertise.

I am simply advocating for thoughtful discernment and the understanding that you will experience the impact of your choices. So take feedback that you need in order to strengthen your systems thinking insights ... and leave the rest.

One final note on the golden rule: It does not only describe a single instance, it can also describe a pattern. If someone you rely on for feedback only gives you critical feedback, no matter what you improve, that person's feedback is inherently unreliable regardless of their level of expertise. They can't, as they say, see the forest or the trees. Find someone who can.

Include Learning

Imagine you recommend a change and all the feedback you get confirms that it's the best idea ever. Three hours after your recommended change is pushed to production ... chaos ensues. It was, it turns out, a terrible idea that nobody predicted correctly.

Conversely, imagine you prototype an idea that everyone said won't work. Low and behold, it works! After it's pushed to production, the company generates unexpected revenue and everyone forgets they said it was a bad idea.

Imagine you push a change to production and hackers figure out a way to exploit it. Or users do something totally unexpected with it.

Regardless of how sound our reasoning is and how expertly-designed our feedback loops are ... we don't know what will happen. How will you know that your recommendation had the intended impact? How will you measure that impact? What are you looking to see change? What are you trying not to change?

When you design a feedback loop, extend the feedback you get past the decision-made point. Include methods for measuring the idea once it becomes a living, breathing change in the

world. Learn from your experience and apply that learning to subsequent thinking. That's, fundamentally, the core systems thinking practice.

Four Core Skills

Whenever two or more people engage in systems thinking and feedback together, there are four steps involved.

1. Listening
2. Thinking and considering
3. Engaging with the reasons
4. Looking for logical fallacies

You might imagine you already know how to do those obvious things. No doubt you do! But we can also improve them as our knowledge worker skills improve.

How to Listen

Healthy feedback loops depend on how well we listen to each other. When engaging with someone else's recommendation, we deeply consider other people's point of view and look for ways to affirm and acknowledge it. This means challenging our

common technology-culture habit of listening for what's wrong. Or listening with a “change my mind” mindset.

Listening is an active activity. We lean in and listen with self awareness. We notice what happens in our own minds, reactions and responses, and we also seek to understand what is happening in other people’s minds. How did they come to this conclusion?

Someone has recommended a change to the API structure in a software system. To indicate you are listening well, your first response might be: “I appreciate your thinking about this API change, I agree it’s important.”

Respectfully acknowledge what you’ve heard. Repeat back your summarized understanding, so other people have an opportunity to correct it. “What I understand is that the current database to database communication is problematic for a number of reasons and you want to shift the two software parts to communicating via an API?”

Listening does not mean you agree with what you hear. Listening is learning and it’s building trust in systemic reasoning. We listen as part of knowledge work and we can learn to demonstrate that we are, in fact, listening.

Change Your Own Mind

When we think together, we are willing to change our own minds. We are all willing to see things differently. We are especially willing to hold ambiguity. There is often no single, concrete answer to systemic challenges. Our thinking is usually partly right and partly wrong. We can work together to build well-reasoned theories about what is best to do, under the circumstances. Just like we work together to build well-structured software.

Giving and receiving feedback is a process of investigating your own mental models. Do they hold up? Do they need deeper consideration? When you engage in feedback loops, you are willing to learn and grow and change. If you aren't, you aren't likely to be giving worthwhile feedback. You are also missing out on the fun of systems thinking, which is learning together how to improve the systems we are building.

In our API recommendation example, you no doubt have opinions about how APIs should be implemented. The goal is to discern if those opinions are relevant, valuable and well-reasoned under these circumstances. If not, be willing to change them.

Engage with the Reasons

Can you add or contribute opportunities to strengthen the reasons? This is what you are listening for, helpful ways to engage with the reasons. In response to the API example:

- “My understanding of the data structure is slightly different, may I show you where I think the labels need changing?”
- “I’m concerned about performance, let’s model how it will handle critical load, shall we?”
- “This approach was taken because, at the time, there was a blocker to securing access roles in our current stack. Can we walk through how that will work here? I’m not sure it’s resolved.”
- “Another reason that supports this approach is ...”

It is relatively easy to dismiss ideas that do not agree with yours. Systems thinking is developing the ability to share why you think differently, how did you reach a different conclusion? And sharing that perspective. It is also about noticing flaws in reasoning.

Look for Fallacies

*He who establishes his argument by noise and command
shows that his reason is weak.*

—Michel de Montaigne, French Essayist

A logical fallacy is a flaw in reasoning. The further we move away from linear, incremental reasoning, the more space we create for logical fallacies to flourish. Like weeds in a vegetable garden, logical fallacies will arise and you'll need to be vigilant about pulling them out.

Erroneous leaps of logic trigger a spiral of fallacious thinking. If you pay close attention, you'll see that when people share logical fallacies, rather than sound recommendations, they trigger a maelstrom of persuasion politics. In other words, people freak out but nothing productive comes of it. Fallacies are inherently emotional and make it difficult to stay on track. You can't respond to fallacies with logic because once they take root, the logic is broken.

Advertising and politics *depend* on the use of logical fallacies to control discourse. We are exposed to them constantly. A politician tells you what's wrong with the opponent, rather than making a case for themselves. Beauty product ads make us afraid to grow older. Ads that begin with "Everybody knows

that ...” are always fallacious. There is nothing that everybody knows. Movie trailers touting “from the producer of A Great Movie You’ve Probably Liked” aren’t giving you any evidence that *this* movie is great.

Fallacies are powerful, they appeal to our baser instincts and feel important for convincing people.

NOTE

We are trained to use logical fallacies rather than to spot them. Systems thinking is learning to spot them.

Bugs in our reasoning will inevitably lead to poor systems design. They are a form of reactive reasoning. As you can see from the following examples, logical fallacies influence our software systems thinking in ways that are so familiar, you might not be aware of the negative impact they generate.

Strawman

Hollis recommends offloading complexity by using cloud-native tools. Briar responds: “Hollis hates open source.”

Briar has shifted the discussion away from the benefits and drawbacks of cloud-native tools. “Hates open source” is an

exaggerated misrepresentation of Hollis' view, and it's beside the point.

Instead, Briar could help Hollis consider the benefits of open-source alternatives and compare them to the benefits of cloud-native tools. “Have you explored open-source alternatives? What are the drawbacks?”

Anecdotal

Hollis recommends building a new capability using microservices, with detailed reasons supporting the recommendation. Briar says: “We tried microservices and it was a disaster.”

Briar is using personal experience and an isolated example to dismiss, instead of engage, Hollis' reasoning. As they say in finance, “Past performance does not predict future results.” The insights gained last time might improve the odds of success this time.

Either way, Briar needs to demonstrate why past experience is relevant now.

Appeal to Authority

Hollis recommends building a new capability using microservices, with detailed reasons to support it. Briar says: “Our CTO said that microservices won’t work for us.”

The fact that an authority thinks that something is true ... does not make it true. This isn’t a sufficiently-valid response. What were the reasons supporting the CTO’s conclusions?

A better response would be “The CTO is concerned about X, Y and Z in our circumstance. Can we discuss those concerns?”

Bandwagon

Hollis says: “We need Kubernetes, everyone is using it now.”

Popularity is not validation. Hollis needs to show why Kubernetes is a valuable priority in this context right now.

Black or White

“If you can’t pass this whiteboard test, you can’t succeed at this job.”

Be wary whenever anyone argues a binary reality ... the world is full of possibilities. Unless your entire job involves passing whiteboard tests in stressful interview scenarios, this is a

dubious claim. A famous American example of this fallacy is the New Hampshire state motto: Live free or die. Maybe, move to a different state?

Middle Ground

Product team A says the organization needs a car. Product team B says the organization needs a boat. So, the organization builds a carboat.

A compromise between two extremes is not always the truth ... or the best solution. Compromise can be a sneaky way of avoiding difficult, collaborative decisions. What does the organization most need? (Hint: It wasn't a carboat.)

Burden of Proof

In response to a recommendation shared by Briar in a Google Doc, Hollis commented: “This looks like a graph and graphs don’t scale.”

We see this fallacy so often, it might feel normal. The comment shifts the burden of proof onto the recommender. Hollis needs to say more. What reasons justify the claim that graphs don’t scale? How are those reasons directly correlated to the context and reasoning in Briar’s recommendation?

Ad hominem

Hollis makes a compelling recommendation for adopting event-based interactions. Briar says “Hollis overcomplicates everything!”

Attacking someone’s character in an attempt to undermine their argument does not undermine their argument. It avoids their argument. (And it’s mean.) Even if Hollis habitually overcomplicates things, Briar still needs to describe how *this* recommendation is overly complicated.

Appeal to Emotion

An architect shares a recommendation with the engineering team and asks for feedback. The lead engineer tells the team not to respond because “architecture is a bullshit role that doesn’t respect our expertise!”

This is an emotionally-loaded response meant to influence the feelings and actions of the team. This scenario is so common, you might be thinking, “hell yes, the lead engineer is right!” Or, at least, you might imagine a scenario in which the architect deserves this reaction.

Doesn't matter. This is a fallacious response because it has nothing to do with the recommendation's validity. Architecture might be a bullshit role and the recommendation might still be valid and worthwhile.

Logical fallacies seem effective but they rarely get us what we really want. The lead engineer wants more respect for their expertise ... Does that reaction earn the team more respect? Unlikely..

Slippery Slope

“If we host parts of our system in AWS, we will be locked in and paying exorbitant fees.”

You can not negate a recommendation simply by predicting catastrophe. If there is a connection between one microservices hosted in AWS and bankruptcy, you need to explicitly describe it.

Summary

When you take a systems view, there is too much complexity to rely only on what you already know. You need to learn from other people's experience and expertise. More importantly, we

need to build conceptual bridges that cross our gaps in the way we understand and experience the system. We need a supportive system of feedback loops that strengthen our systems thinking and enable us to reengineer our current mental models, together.

Designing feedback loops that improve our systems thinking includes asking for the feedback we need, from the people who can give it to us. It is discerning what feedback to take and, when we act on our thinking, learning from it.

When we are part of a feedback loop, we are critically listening, practicing metacognition, engaging with the reasons that support our conclusions and looking for logical fallacies. Logical fallacies flourish in our conceptual gaps and everyone benefits from knowing how to spot them.

Practices

Using work from the previous chapter, create a recommendation and strengthen the reasons. Then design feedback loops:

- What feedback do you need?
- Who might give it to you?

- How will you measure the impact and learn from this decision?

Once you collect this feedback, consider what you will take and what you will leave. Do you need to gather more?

Use the feedback to improve your recommendation.

Also, offer to give feedback to someone else. Practice the core principles of engaged listening, noticing your own mind and being willing to change it, strengthening the reasons (if you can) and noticing logical fallacies.

Questions for Your Journal

- What is the hardest part about building conceptual bridges? Where has that work failed in your experience?
- What has my experience been with feedback loops and how can I improve it?
- What are some examples of helpful feedback I have received?
- What are some examples of unhelpful feedback I have received?
- Review the logical fallacy examples in this chapter and write some examples that you have heard.

“Goldberg is best known for his popular cartoons depicting complicated gadgets performing simple tasks in indirect, convoluted ways.”

https://en.wikipedia.org/wiki/Rube_Goldberg

Read Andrew Harmel-Law's upcoming book *Facilitating Software Architecture*.

Chapter 9. Pattern Thinking

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author’s raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 9th chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at sevans@oreilly.com.

“No pattern is an isolated entity. Each pattern can exist in the world, only to the extent that is supported by other patterns: the larger patterns in which it is embedded, the patterns of the same size that surround it, and the smaller patterns which are embedded in it.”

—Christopher W. Alexander, *A Pattern Language: Towns, Buildings, Construction*

I confess, I am a pattern thinker by nature. When I catch a cold, I think about the factors that contributed to getting sick. Have I been stressed or overtired? Was I exposed to groups of people or someone sick? I compare my symptoms to colds I've had in the past, and to not-colds like seasonal allergies or Covid. I research, and by “research” I mean google. Are there new, better treatments available? Would my symptoms benefit from a doctor’s visit? What has helped me get well in the past? I alter my schedule to accommodate a few days of recovery.

Then ... I try to ignore whatever support plan I created. “Just power through!” is my long-standing reactive pattern to getting sick, despite being fully aware of the inevitable outcome. Like that Friday afternoon, early in my software engineering career, when I was brain-foggingly sick and didn’t want to admit it. I pushed a small Javascript change to production. On Monday, I

discovered that the e-commerce process (which I wasn't changing) was broken all weekend. Nobody could purchase items in their shopping carts.

(This was before test coverage was a common practice.)

Fortunately, the site was relatively new and traffic on the weekend was still low, so I learned my lesson without costing a company millions of dollars. Still, the loss of one sale was big for my client. Being sick impacts my thinking. Nowadays, being pattern aware helps me (usually) recognize my impulse to act heroic and go to bed instead.

I've been accused of "overthinking" things. Sure, in some cases, that's a fair assessment. I was exposed to a virus and caught a cold. If I wasn't exposed, I wouldn't have caught a cold. If I rest, I'll get better. Nuff said.

Yet, if we look deeper at the behavior of viruses in humans, we see that what happens in a person's physical system after exposure depends on other factors. Physical condition and diet, for example. I had chronic respiratory illnesses that would often cascade into infections like bronchitis. I discovered (through pattern thinking) that some foods were wreaking havoc with my immune system. I changed my diet and then, I

was rarely sick. At the time, doctors were not looking for those types of patterns.

Characteristics of the virus plays a role. Your body behaves differently if you've been exposed to that virus before. Sometimes. Outcomes can be unpredictable. Not everyone exposed to Covid, for example, gets sick. A leading expert, who happened to be waiting in line with me at the Reykjavik airport, said that there aren't just superspreader events – there are superspreader people. Some of us exposed to Covid will transmit it to nearly everyone we come in contact with. Some of us get Covid and no one else gets sick. We don't know why this happens or even if, by the time you read this, it will still be how we understand transmission patterns. Experts are watching patterns and some of those patterns are surprising ... we don't yet fully understand what we are seeing.

The application of complex systems theory to viral dynamics has provided new insights into the development of AIDS in patients infected with HIV-1, the emergence of new antigenic variants of the influenza A virus, and other cutting-edge advances.

—Ricard Solé and Santiago F. Elena, Viruses as
Complex Adaptive Systems

This is life in systems. The closer you look, the more complexity you discover.

In This Chapter

In this chapter, you will learn what pattern thinking means and how relationships produce effect. We'll explore where to look for patterns and use our example company, Mago, to explore common patterns. Systemic leadership is encouraging pattern thinking when solving challenging problems.

The concepts we will explore include:

- Pattern thinking is not simply learning patterns you can apply. It is learning to discover, discern and describe patterns in your circumstances.
- Counterintuitiveness will always be involved in pattern thinking.
- Pattern thinking is exploring how relationships produce effect.
- The same event can arise from very different patterns.
- You look for patterns in relationship to time, events, context and structures. You look for patterns in your own mind.
- There are always external, technological and process patterns involved.

- As software and information become ubiquitous, many organizations are facing paradigm-shifting pattern changes.

The practices you'll discover are:

- Read some resources on pattern thinking.
- Using “where to look for patterns” guidelines, explore patterns in your current circumstances.
- Consider patterns in the Mago situation, are they familiar to you?

What is Pattern Thinking?

Systems thinking often involves moving from observing events or data, to identifying patterns of behavior over time, to surfacing the underlying structures that drive those events and patterns.

— Michael Goodman¹

Like most phrases in systems thinking, there are multiple definitions of pattern thinking. When we look at a pattern, we are mentally isolating it from the complexity in which it operates. How we define pattern thinking depends on our perspective, the frame through which we look at patterns.

In software, interest in patterns was popularized by books like *Design Patterns* (1994) and *Pattern-Oriented Software Architecture* (1996). We have framed the word “patterns” to describe object-oriented approaches to software development, like the factory pattern or state. We continued to evolve our thinking about patterns, describing relational patterns in software, like client-server and event bus. Enterprise integration patterns² arose with software system complexity. Interface development using design systems and components is, fundamentally, applying pattern thinking to create reusability and consistent user experience.

When we talk about DevOps, stream-aligned teams and microservices architecture, we are talking about patterns.

In systems, we expand the scope of our definition – we use bigger or different frames. System patterns depend on the circumstances, so there are fewer generic patterns we can apply and significantly more need to observe how the system operates before changing patterns. Patterns in our software systems are always evolving. I will give you pattern examples, using our fictitious Mago system, introduced in Chapter 3, which is facing a profound pattern-change problem. You will likely recognize some of them. But I guarantee that you are also facing pattern challenges that are different from Mago’s.

NOTE

Pattern thinking is not simply learning patterns you can apply. It is learning to discover, discern and describe patterns in your circumstances. And it is learning to transform patterns to steer a system in a different direction.

A common definition of pattern thinking is to consider whether or not an event has happened before and whether or not the circumstances are similar. But be careful: if you imagine patterns only along a linear timeline, you will stay entrenched in linear thinking even as you apply systems-thinking approaches. Patterns that repeat are also, often subtly, changing as they repeat. When we are able to spot the repeating patterns AND understand the ways they are reinforced as they change, we are pattern thinking.

To illustrate, let's use the “there's a bug in production” scenario again. In this case, the bug is a syntax error that a linter would have caught. Your team has been extra busy, pushing changes faster than usual. You are missing little things because you have less time for peer review. You decide to add a linting step to the merge process and, for a while, this decreases the bugs.

This is an example of pattern thinking: rather than view each bug as its own discrete event, and react to it, you see the pattern – a repeating event with a common feature (syntax errors). You

see that the circumstances are encouraging it. You intervene (prevent or alter the course of events) with a relatively-small change that improves the system as a whole (by keeping it tidy with linting).

Here, though, is the no man's land between linear and nonlinear: Does linting change the root cause of the problem?

Perhaps. There is nothing wrong with adding a linter and calling it good. We don't need to overthink everything, all the time, and engage in root cause analysis. Many teams use linters because they are helpful. A valid concern about systems thinking is that it adds conceptual load onto people who are already at capacity. There's no need to overdo it. You can, however, look deeper into the situation and see what you see.

Remember the iceberg model (introduced in Chapter 3)? Underneath events are patterns and looking at them is a great first step towards systems thinking. Underneath patterns are structures ... forces acting on patterns to cause or reinforce them. Structures arise from our mental models and, as we've said previously, if you don't transform the mental models, you won't effectively impact the system as a whole.

Why is the team pushing code to production faster? Why has their peer review process become strained and sometimes ineffective? Why wasn't linting already part of the delivery process, was there a reason?

Perhaps the team was ready to move faster and syntax errors in production simply reflect that they've been overcautious. They were perfectionistic in their approach, unwilling to accept that it's perfectly normal to miss errors in code. We all want to believe that we can write perfect code, play perfect music, write perfect books, recommend perfect architectural solutions, give perfect Ted Talks but the mind doesn't work that way. When we are generating something novel, we miss things. Everyone does and that's unavoidable. A linter might be the safety net that allowed the team to take more risks and increase their pace.

Perhaps they had a manager who was reinforcing this perfectionistic structure. The manager's core mental model was "good developers write perfect code quickly". Patterns of blaming and derisive feedback loops were slowing the team down. They were rarely pushing bugs but they were also rarely pushing code.

Now they have a new manager, who encourages them to stretch more and try harder things. They have more bugs because they

also write more code. At first, they make more mistakes but over time, they make fewer. A linter supports them as they grow. Perhaps they are also improving test coverage, supporting the change in multiple ways. From a systems point of view, for the moment, I would call the root problem solved.

Perhaps the team is under pressure to deliver faster. In my experience, this is the most-common cause and not always a bad thing. Teams have crunch times when they are down in the weeds together trying to hit a launch deadline. Sometimes, despite their best efforts, things get messier than usual. They might tolerate a leftover TODO or three that they wouldn't under normal operating conditions. Once the crunch time passes, they tidy up. Meanwhile, they add linting.

In this case, their structures and mental models are flexible enough to adapt to circumstances. Overall, they do excellent work and they know that sometimes it's harder to find low-level errors than other times. Their work over time is trustworthy so they don't worry too much about the crazy days.

More often though, the pressure isn't the natural rise and fall of activity in software development. When I ask workshop attendees to identify the core mental models and structures that lie beneath bugs in production, the most common theme is fear.

An organization, for one reason or another, is reinforcing their own chaos. Some examples include:

- Believing they are a ship at sea during a storm. Constantly fearing they might sink, they drive everyone to row faster.
- Believing that they can only gain competitive advantage, be first to market, through sustained authoritarian leadership, bullies are promoted and rewarded.
- C-levels are constantly fighting each other for control, which translates into constant drama and distraction for the teams. People begin their week working on one thing, only to be told they also need to work on three other Priority One issues simultaneously. Without missing their deadlines.
- The frailty in the software system has, over time, created an “always on fire” culture that has become the norm.

A sustained core mental model of “Deliver quickly or we die” will rarely deliver anything faster over time. Nobody thinks well when they are constantly on fire. When delivering as fast as possible is the only goal, a syntax-error bug in production acts like oil on the fire of fear that’s already raging. More pressure is put on the team – the bugs are perceived as their fault, not the natural outcome of fearful mental models.

Adding a linter won't solve this core problem. It's just a bandaid. What happens next? Harder-to-spot "bugs" in the system that arise from poorly architected relationships, burn out in the team, products that are quickly brought to market but untested by users (who are unpredictable at the best of times). More managerial oversight might be added, increasing the conflicting expectations on the team.

This is a vicious cycle I've seen time and time again. It describes, in a fundamental way, why I am writing this book. Whenever we make changes in a system ... The impact of the change depends on *why* we are making that change. What are the mental models and reinforcing structures involved? The efficacy of our changes depends on how well we understand the patterns that are acting on the system and how well our change addresses core mental models.

I got tired of never-ending solutions that don't solve the core problems. Perhaps you've felt this too? As relational complexity increases, our command and control approaches become more noise than signal. Pattern thinking can help us amplify signal rather than amp up the noise.

Counterintuitiveness, Again

This is a good time to remind ourselves of counterintuitiveness. As defined in Chapter 2, and mentioned in subsequent chapters, counterintuitiveness is when the “right” thing to do in a system seems wrong to us. Many pattern changes will be counterintuitive. We are swimming in familiar patterns and like a fish in water, they feel like our natural habitat. While I was writing this chapter, I collided with my own counterintuitiveness. I’ll share the story, even though it isn’t about a coding project, to illustrate how sneaky counterintuitiveness can be.

For many years, I’ve gotten up every morning, made coffee, and written (by hand) in a journal. You know this already because I wrote about it in Chapter 4, where I encouraged you to do the same. (I still encourage you to do this.) I’m not great at consistent habits, practicing does not come easy. I teach the value of practices because I know from bitter experience how valuable they are. My morning writing practice took me decades to develop, but over time, the practice stuck. It’s one thing in my life that just works.

Last year, when I began writing this book, I took a three-month sabbatical from my architecture role. For the first six weeks, I unplugged from everything, minimized my household responsibilities, and wrote. In the evening, I read. It was

luxurious. I wrote tens of thousands of words and learned thought-provoking things about systems (human and otherwise) in my reading.

Halfway through my sabbatical, I joined a few, low key meetings and started answering emails. My writing practice evaporated, a river of words became a puddle. I struggled to show up to writing sessions and my output reduced to 10%. My first-thing-in-the morning journaling practice remained my refuge. I could show up there. But despite the pressure I was putting on myself, transitioning to my desk, where I'd write the book, was always difficult. I'd procrastinate. And that procrastination threw off everything else I needed to do during the day.

After my sabbatical, the paradigm around me shifted. Deadlines for workshops, talks, and other projects became unprecedently nonstop. I began traveling internationally 25% of the time. I left my role and started working on a greenfield initiative in a new-to-me technology stack. My life was in flux like never before. I wrote that systems thinking is about becoming comfortable with uncertainty. I felt tested by that assertion.

Still, I kept working on the book. You can probably see what's coming ... I started to struggle. I was heading for the iceberg. My habit of pressuring myself to get more done was ineffectual at scale. Ironic, isn't it?

I was barely keeping my head above water. My morning journal was filled with stress and failed strategies for getting through the day. Anxiety about writing increased in me until it was screaming so loud, I had to take it seriously. I signed up for an immersive writing workshop with a teacher who has published 50+ nonfiction books. He has coached people for 40 years, so he's seen it all.

He said: "Write every morning, first thing, every day."

"Great," I said, "I already do that!"

"On your book," he said, "move your morning journaling."

WHAT?!?!

I saw my morning writing practice as sacred ground, hard won and worth defending against all encroachment. I didn't see it as what it might be – years of training myself to get up and write.

Monday morning after the workshop, I got up, made coffee, sat at my desk and opened this chapter. As I type this, it's 6:30 am, out the window I can see the sun is rising over the hills and frost on the grass. One of my dogs is sleeping on the chair next to me and I'm nearly done with my book writing sessions for today.

When I was no longer stressed about making space in my day for the book, I felt grounded in incremental progress by breakfast. It lessened the number of time-management decisions I was making in my already busy days. I slept better. I flowed through the day easier. I made space for the other deadlines to be met more comfortably.

I knew I'd made the correct change, for now, because the system as a whole (in this case, my personal operating system) improved its ability to serve its purpose (my work). I was also, overall, happier. Which is a big surprise, given that I thought my morning journaling was responsible for much of my happiness.

Here's the important point of this story: Because of counterintuitiveness, I couldn't know whether or not shifting from my journal to the book first thing would have a positive impact. I had to *actually try it* and see if the teacher was right.

He was. I had changed absolutely everything except the one thing that would really make a difference. I was trying to do things the way I'd always done them despite a shift in paradigm. In that new paradigm, my brain needed structures that were different from the ones I usually rely on in order to adapt to changing patterns.

It's amazing to me that I could be blind to my own teachings. Luckily for me, when you write a book, it teaches you.

Today, you are blocked by counterintuitiveness too. That's okay. Even if you think it's not okay, it doesn't matter, it's still gonna block you. We dance with our circumstances until we can see that one small change, often the one we most don't want to make, will unlock potential. That change will rarely be the first idea that occurs to you. You just have to listen to the noise, feel the painfully stuck spots and keep looking until you see it. Or, until some random person in a workshop happens to point it out.

A single change in a daily pattern changes my relationship with other aspects of my life. This is because, when it comes to patterns, we are watching how relationships produce effect.

How Relationships Produce Effect

We can't impose our will upon a system. We can listen to what the system tells us, and discover how its properties and our values can work together to bring forth something much better than could ever be produced by our will alone.

We can't control systems or figure them out. But we can dance with them!

—Donella Meadows³

Patterns are the choreography of a system. Everything is in relationship to everything else and patterns we don't explicitly design will emerge, despite our best efforts to control them. Behaviors in systems arise from relationships between parts and the way those parts share information. Structures, networks of relationships that create behavior, reinforce thinking and behaviors, in software systems and in people.

Pattern thinking is looking beneath the surface of events to discern how relationships produce effect. Relational patterns are, perhaps, more familiar to us outside of technology. You, as an individual, have patterns of thinking and behavior. When you get into a relationship, you, as a couple, have patterns of thinking and behavior that neither of you exhibit alone.

Patterns you might discuss in couples therapy. When you have

children, the relational patterns have an increasingly-complex and long-term impact. Psychologists call this a “family system” because familial relationships produce effect beyond what any one person says or does.

Not just direct effect, but indirect as well. A family system is in relationship to external systems, like employers, grandparents, schools, religious beliefs and the communities in which the family lives. All of these relationships will influence the way a family thinks and behaves. When you can't sleep because you are feeling stressed about work and family pressures, you are experiencing how relationships, in organizations and in our homes, produce effect.

The same is true for software systems. Pattern thinking is understanding how the system of software, the people building it, and the organizations around them, produce effect. Pattern thinking is also discerning ways to improve patterns in order to improve the system as a whole.

This is a messy subject to explore because each software system will have unique patterns. And each software system will share common patterns with other similar software systems. At the intersection of these unique and shared patterns is architecture

– the process of figuring out how to dance with this particular system.

When we shift into pattern thinking, we move away from demanding concrete answers to complex challenges and into the whitewater world of interrelationships. When you understand patterns, you understand that everything is in flux . Like a sailor, we learn to navigate towards our destination in respectful relationship with the forces acting on us. We can't ignore the ocean, the weather or the need to provision regularly. We can discern how to operate wisely in whatever circumstances we find ourselves in.

Modeling Patterns

System patterns aren't usually visible at the code level. Modeling is essential for pattern thinking. Modeling together makes patterns visible and synthesizes disparate views about what happens in a system. I always use a whiteboard (digital or physical) to facilitate discussions about patterns. Collective modeling shifts people away from their usual mindset into the world of bounded shapes and interrelationships.

Frameworks help structure our code and our thinking about a system. Software frameworks, like Spring Boot, Symphony or

Angular, save us design time by enforcing patterns. We model technology systems using enterprise architecture frameworks, like TOGAF or Zachman, and software modeling frameworks like C4. We use tools like UML or ArchiMate to show systemic relationships.

In my experience, wholesale adoption of any one framework is insufficient to support systems thinking. Which doesn't mean we shouldn't use them. While there is no one modeling approach that will help you understand every system, you can learn a lot by using whatever frameworks or tools help structure a more-comprehensive understanding of your situation. You can mix business process models with UML diagrams, domain models with pictures of messy whiteboards made during a discussion. In models, like in systems, there is no One Right Way, no silver bullet. You are always synthesizing other people's expertise and experience while inventing the practices that will support your situation.

We more-often draw on metaphors to describe patterns, as I do in this chapter. This can be frustrating for those of us who equate metaphor with “too abstract”. I understand that frustration. Modeling practice helps us use “just enough” abstraction to make patterns visible and sufficient data-driven observations to ensure we are describing reality.

In Chapter 10, we'll dive deeper into modeling. Modeling patterns is both art and science, an opportunity to generate insight in both traditional and novel ways. As you read this chapter, I encourage you to imagine how you might model the types of patterns, and exploration questions, I describe.

Same Event; Different Patterns

In the previous “bug in production” example, I showed how the same event can be caused by very different patterns, structures and core mental models. Here is another example of how the same event can be influenced by very different patterns and sociotechnical structures ...

An organization experiences a major, unpredicted outage in a business critical part of the software system. Everyone who can get it back up and running is put on high alert. They triage the problem, figure out its root cause and resolve it. The outage was a rare occurrence, triggered by a cascade of outages in third-party dependencies. There was very little they could do to prevent it, though they recommend some changes to reduce impact if it happens again (which is unlikely).

Conversely, an organization experiences a major, unpredicted outage in a business critical part of the software system for the

32nd time this year. Everyone is on constant high alert and leadership wants to know who is to blame. They put increasing pressure on the development teams to write perfect code in a software system that is increasingly frail and unpredictable.

This is a self-reinforcing pattern but not in the way you might expect. Figuring out the root cause of recurring outages requires deep knowledge and experience with the software system. People aren't staying long enough to develop that expertise – they leave for roles that treat them more respectfully or become reticent to share ideas. Patterns within these outages are clues to the causes, but new developers are re-climbing the same learning curves as their predecessors. The few senior people that do have expertise are burnt out and condescending to new developers, so they are avoided whenever possible.

At the system level, reactive design patterns emerging in the industry would improve the performance of the system as a whole. The current frailty is caused, in large part, by trying to scale software suited for one paradigm into a world that's changed. But the long-term engineers reinforce "this is how we do things here" patterns and structures, strongly resisting changes suggested by people they feel don't understand their problems.

The whole system is dead in the water despite constant drama and activity.

HR is told to find top talent, the elusive 10X developer who can solve their problem ... but top talent recognizes toxic delivery patterns and steers clear. Top talent also want to build with modern tools. Until the organization changes their core mental models, and builds structures that support cooperative evolution, the outages are going to continue regardless of how many patches are pushed to production.

Where to Look For Patterns

Pattern thinking is detective work, watching what is observable and sussing out the (sometimes invisible) connections between events. There are often multiple forces acting on observable events. We use pattern thinking to help us decide which ones have the most impact, under the circumstances.

When there is a bug in production, for example, that event potentially has four important relationships:

In Relationship to Time

Does the event repeat? If so, when? Has the repeating event changed over time? Has the frequency changed? Are the bugs in

different parts of the code but always revealing themselves during a deployment?

Some patterns help us to improve a system over time. Best practices, when done well, organize our thinking, behavior and technology relationships that establish healthy patterns for code quality, delivery and interactions between parts. Some patterns degrade the system over time. We see the symptoms of this decline in what we call “tech debt.”

Time is always a factor in systems thinking. As things change, the value of our established patterns change. The best way of doing things yesterday might be the worst thing to do today. Our past selves have generated challenges for our future selves ... sometimes we are cleaning up a mess and sometimes we are pleasantly surprised.

In Relationship to Context

Context is an understanding of the circumstances. A system serves a purpose in relationship to the context, circumstances that are changing. The purpose of Netflix’s system has always been to provide movies on demand. But imagine the differences between a system designed to mail DVDs to monthly subscribers and one designed to provide streaming movies on

demand. Operating well in the context of distributing movies depends on understanding, and responding to, the ever-changing circumstances.

At the software level, we are used to thinking about context. Does an event happen in one context but not in others? A bug that only appears in Safari indicates a relational problem between the source code and the browser.

On a systems level, leverage points, the best changes to make in a system, completely depend on the context. What works for Netflix or Spotify or Facebook won't work for any other company, at least not in the same way they worked for Netflix, Spotify and Facebook. The patterns in each circumstance will be, at least somewhat, different. What works for Netflix, Spotify or Facebook today might not work for them next year.

What we prioritize depends on context. Some software systems need to be fast; some need to be perfectly accurate; others need to be exceptionally easy to change. We might want all three, all the time, but we prioritize which qualities matter most based on our context. A kinda-slow query isn't always problematic, why does response speed matter in these circumstances?

In Relationship to Other Parts

Decoupling and modernization have become common phrases over the last 10 years. These are pattern changes that happen, primarily, in the relationships between the parts. You'll find many resources that help you understand API design, event-driven interactions, microsites and microservices. Conferences and talks increasingly include Domain-Driven Design⁴ and Wardley Mapping⁵ topics to help us design parts of a software system, and their relationship, to match the context (the domain) in which we are building them. We are encouraged to use domain language to describe technology parts and interactions. Popular books like *[Team Topologies](#)* and *[Accelerate](#)* integrate people relationship design with technology design.

Despite all this attention to relationships between parts, I rarely see project or program management approaches that support emergent systems design. By emergent, I mean the whole becomes greater than the sum of its parts. Generally, we “manage” by breaking down the parts and adding control structures, like SaFE or concrete delivery life cycles, rather than orchestrating interrelated but somewhat-independent activity.

Control structures are partly necessary, we need to coordinate what an individual works on today with the overall goal of the system. We hit the iceberg though when we try to design a system inside linear-thinking approaches. We also need

systems-thinking approaches that welcome uncertainty as a partner in design and encourages innovation by creating space for learning together. We need relationships, in the people and the system, that are, whenever possible, self organizing and deeply aware of context.

My experience is anecdotal. I'm sure there are groups that have figured out how to apply "just enough" control in the midst of their system design and delivery process. We need to amplify that thinking, wherever we find it, so that we can improve the way we structure relationships between people and technology parts in the systems age.

In Relationship to Structures

A pattern is always held in place by structures of thinking, behavior, information sharing etc. Sociotechnical structures reinforce patterns and keep them from changing. Hierarchical leadership is a structure. Delivery methods and project management processes are a structure. Test-driven development is a structure. Shared beliefs about the ways we should communicate thinking (or not) is a structure.

In the technology system, there is, of course, infrastructure. Infrastructure is deeply involved in establishing and governing

patterns in a technology system. It can also be a leading source of hard-to-find-and-fix systemic issues. In systems I've worked on, for example, the caching structure was both keeping the system running efficiently and the leading cause of unpredictability.

All change begins with a decision and decision making is always interrelated to structure. Here is an example:

Developers working in a legacy system want to build test coverage so they can decrease bugs in production. They invest too much time, they say, fixing problems when their code inadvertently impacts code elsewhere in their Big Ball of Mud codebase. A short term investment in tests will pay off in the long term.

But decisions about where the team invests their time are made by product managers who are pressured by leadership to deliver more change faster. Improving the system is rarely prioritized because each initiative has a budget that only covers new development. The organizational decision-making structure leaves no space to resolve these competing needs.

Remember the carboat example from Chapter 4? One team wants a car, one team wants a boat, so the engineers build a

carboat, which nobody wants? Groups within an organization are pushing and pulling technology decisions in different directions. How we grapple with social forces will inherently architect the system. Few organizations structure teams to operate cross-functionally but that is where the patterns we want to improve usually exist.

In Your Own Thinking

The most important place to look for patterns is in your own thinking. When I was an engineering lead, I helped build linear-thinking approaches that worked for us, at the time. However, as relational complexity increased, I realized that I wasn't practicing sufficient pattern thinking in my work and hadn't been for years.

That's not a bad thing. In the world before "modernization", my teammates and I delivered code that was well-encapsulated by the software's framework. For example, pushing PHP code into an ever-expanding CMS framework didn't require as much pattern thinking as designing an event-driven microservices platform. My need for systems thinking reflected the changes in the world around me.

The pattern thinking that enables me to design software in one context is different from the pattern thinking that enables me to succeed in another. Imperative patterns, setting A equal to B + C differs from reactive patterns, changing the value of A whenever B or C changes. The differences aren't just in the code, they are in the entire structure of the system.

Sometimes, the hardest thing to change in a system is our own thinking.

Three Types of Patterns

As software professionals, we can shift our thinking towards patterns in numerous ways. Another way is to look at the three groups of patterns involved in software development.

External Patterns

Software is being influenced by patterns that exist beyond the boundary of our software-building experience. We tend to imagine that our users have static needs that we can understand and design towards. This is somewhat true. But those needs are constantly being influenced by experience they are having in the world outside of our domain.

There was a time, for example, when the user interface could be scrunched and boxy, in frames, with limited fonts and perhaps even an animated gif. Now users would view that as an untrustworthy scam. They expect white space, responsiveness, variation of visual shapes and high-resolution images. 15 years ago, a user might wait 2 minutes, perhaps even longer, for something to load. Now they'd be long gone. Expectations change as users experience emergent patterns that weren't necessarily what any one company determined.

We build software that fits a paradigm and then, the paradigm shifts. When that happens, our software thinking is especially vulnerable to misunderstanding what matters to change. We need more than modern infrastructure tools, we need to think about the mission, purpose and patterns impacted by external changes.

External patterns can also be social forces that reinforce (or not) hiring practices, the way we define leadership and follow “authority”, who gets money to develop things (and who doesn’t) and how we are all taught software skills in education. The social culture at technology conferences influences the way teams build software and vice versa. External trends, like DevOps practices, interrelate with our internal patterns.

Patterns in the Technology System

Patterns govern where we put new code. Layers are patterns – the application layer is where we put software logic and the presentation layer is where we put look-and-feel logic. We use patterns to make our code reusable. We design patterns that structure parts in relationship, services, for example. Patterns are formed by the relationships between those parts, when and how and what they communicate to each other. Patterns are formed by when and how and what they don't communicate to each other. Nearly everything related to data management follows some agreed-upon patterns of queries and storage.

There are many, many resources for learning about patterns in software, microservices, serverless, etc. As our concern here is systems and specifically, systems thinking, I won't say much more about internal software development patterns. But I will recommend resources at the end of this chapter, if you'd like to dive in.

My personal favorite book on pattern thinking, and the book that many systems architects geek out about, is *A Pattern Language* by Christopher Alexander. This book has nothing to do with software. Yet Alexander triggered the pattern language movement in computer science, which led to changes in Object-

Oriented Programming and agile development. Another favorite is Design Unbound: Designing for Emergence in a Whitewater World by Ann M. Pendleton-Jullian and John Seely Brown. Dr. Pendleton-Jullian has taught at MIT and Stanford and writes about architecture – of literal buildings, not software – yet her thinking reflects many of the challenges we face. Your systems thinking practice will be well supported by learning about patterns beyond software.

Process Patterns

As I've said in previous chapters, systems are sociotechnical. We can see this clearly in our day-to-day experience of patterns. What is the process governing delivery? How are decisions made? How are roles defined and bounded? What is the definition of "done"? What tools and structures do people use to communicate?

Nowadays, I see thinking divided into "product" and "tech" in ways it wasn't early in my career. I've seen waterfall decisions trickle down from leadership and self-organizing, cross-functional teams architecting their own software. I've seen weekly releases and continuous deployment. Groups that help and support each other; groups that despise and sabotage each other.

If you model how money is allocated to technology initiatives in an organization, you will learn a lot about why the system is how it is. Much of what you experience every day will be related to how money flows.

All of the human aspects of software and systems development are subject to patterns that we can (potentially) consider and improve. This book focuses a lot on process patterns not because they are the most important, necessarily ... but they are the ones that will need to change if we want to change other types of patterns.

When we are thinking in systems, are we thinking about external patterns, internal (to the software) patterns or process patterns? Yes! Systems thinking, for us as software professionals, is thinking about how these three types of patterns intersect and interact so we can discern where to intervene.

Discernment is the key word. You can't change all patterns at once ... you wouldn't want to but even if you did, you wouldn't succeed. Patterns exist as a confluence of thinking, behaviors, mental models and long-standing socially-conditioned structures. They rarely change easily. When we think in

patterns, we are looking for small changes that will have a big impact.

But first, we need to see the patterns that are relevant to our challenges.

Seven Pattern Thinking Questions

When you are trying to identify patterns in a software system, here are some good questions to explore, model and consider:

- **How does information flow?** How and where is it created, shared, stored and shaped? If it's in motion, is it transformed? Does the information change as the context changes? Is this flow monitored? Should it be?
- **What are the events** that happen in the system? When one thing happens – what happens next? What core activities define the system's purpose? When do these activities happen and what do they change? (In my experience, changing patterns related to events is an impactful place to intervene.)
- **What are the boundaries** in the system? What are the encapsulated parts and why do they exist? Do they mirror the capabilities of the domain? How strong (or not) are

these boundaries and how strong (or not) are the relationships between them?

- **What are the building blocks** in the system? Are there components, widgets, modules, classes or other ways to reuse logic? How do they interrelate and form structures?
- **What is the delivery process?** How are priorities set? Quality ensured? How are decisions about what matters and what doesn't get made? What are the core mental models and structures supporting this process?
- **How are people organized?** What are the hiring practices? How are teams structured? What groups exist in the organization and how do they interact with each other? What are the practices expected from a manager? How is leadership defined?
- **How is discourse structured?** Who is involved in decision making? Are reasons given for decisions? Does information flow across teams? Up and down the organizational hierarchy? Do teams communicate as peers? Do people listen to each other?

When there are blockers and stuck places that you can't see, because of counterintuitiveness, exploring these questions can often help you see what you are missing.

Mago: Looking at the Patterns

Let's use our Mago example to explore some real world patterns. I introduced Mago way back in Chapter 3, so here is a quick synopsis to remind you.

Mago published the most popular, internationally-distributed magazine in the world. Their state-of-the-art publishing software and distribution system ensured each edition physically reached millions of people worldwide every week. When the internet arose, Mago exported a printed page to make a web page. They built their digital presence using a single piece of extensible software, hosted on-premise. Their data architecture, delivery workflows, and software choreography revolved around the concept of a “page”. Digital page views quickly rose to millions per day.

An entire ecosystem of digital infrastructure followed. Subscription transactions, tracking and analyzing reader behaviors, monitoring software speed and availability, morphing assets (like images) into varying shapes, sizes and types. Business processes digitized, payroll, hiring, project planning. Communication tools became inextricable from

productivity, enabling distributed teams to meet, plan work and track progress.

As the world became increasingly interconnected, Mago's content needed to show up on search engine results, social media platforms, news aggregators, video and audio platforms. Readers no longer interacted with articles by commenting on a page, now they talked about them wherever people chatted online. Mago built an app, then another to share specific types of content.

The shape of information was transmuting and increasingly ubiquitous. The shape of everything was changing.

The single context – a page – became multi-context information that varied. People accessing information on their desktop wanted different information than people on their phone. People wanted bits of information from Alexa or Siri. Mago faced an increasing need to rethink their information structure.

As bandwidth increased, so did the demand for multimedia content. Mago expanded staff to create extended video stories, podcasts, and custom interactive graphics to show data trends. Each of these innovations required software systems (that didn't exist yet) to support them.

In ten short years, Mago's weekly publishing rhythm became asynchronous, 24/7 delivery. More teams needed to keep these workflows in sync. Parts and people were quickly becoming both independent and interdependent.

Even though they aren't a technology company, the Mago teams built a lot of software to keep up with the relentless pace of modernization. When the legacy software became hard to change, product people went rogue and built separate software to support new products. Hosting was moved to the cloud and some processes became automated. Over time, an ad hoc collection of mostly dissociated parts was "glued" together by people moving information by hand.

Very few people could name all the parts of the system or knew how they functioned.

In Chapter 10, we will use models to explore the Mago system in more detail. Here, let's use the recommendations in this chapter to consider patterns.

Patterns in Relationship

In relationship to time, the entire organization had a synchronous structure set up to support weekly delivery of

content. Digital content, at first, stayed within this structure, including a weekly release schedule.

But as the paradigm shifted, development and delivery schedules became asynchronous. As did user engagement, advertising strategies (as they shifted from static to dynamic), product design approaches and team interactions. The system became a patchwork of time-driven processes in silos.

In relationship to context, the mission of Mago didn't change. The system's purpose – to provide captivating and relevant content that people are willing to pay to read - stayed the same. But as the contexts in which that mission was accomplished expanded into many realms, including personalized experiences in the browser or app, their ability to understand and track user behavior in multiple contexts was non-existent. When someone comments on an article shared on social media, for example, Mago doesn't know if that person is also a subscriber or someone who watches videos or someone who reads the newsletter.

In relationship to other parts ... Well, here's where chaos reigns. Software parts, and the teams who build them, and the content creators who need them, came online in a haphazard way. When there was communication between software built in-

house, by vendors and SaaS solutions, rickety duct-taped bridges were built using exports, batch processes or someone just did it by hand. There was so much siloing that no one person knew what all the technology parts involved were. Where there wasn't good encapsulation, the software became a Big Ball of Mud rather than parts in relationship.

In relationship to structure, there was constant intrigue. Budget was given to teams building new products and initiatives that potentially generated return on investment. There was no mental model, yet, for investing in a digital-distribution system. The organization valued “sweating their equity”, making capital investments in technology that would last 10 - 20 years or more, so they were resistant to seeing themselves as a technology innovator. (Even as their position in the industry meant that the technology teams were often ahead of the curve.) They approached digital transformation as yet another project, another capital investment, rather than rethinking their core mental models. When transformation initiatives failed, which they inevitably did, the people leading them were blamed and more “management” was added.

For each individual involved in the sociotechnical system, the internal thinking patterns they'd learned over a decade of working in the old paradigm did not translate well into the new

one. People who did make the conceptual leap left for organizations who were leveraging modern approaches to systems design, event-based interactions, decoupling, continuous deployment. New leadership was hired to drive change but struggled to communicate the change to minds looking for solutions that fit into the current structure of decision making.

External, Technology System and Process Patterns

Over the course of time, these three types of patterns changed. Here are some examples of how they changed.

Table 9-1. External patterns

Then	Now
<p>Information was difficult to get unless you went to the library or bookstore. Timely information was shared on evening news programs, in magazines or newspapers, which had a daily, weekly, monthly rhythm. People waited for it.</p>	<p>Information is ubiquitous and always at your fingertips. Timely information is shared constantly, in streams that readers can't keep up with. Multiple information sources can be scanned and cross-referenced quickly and for free.</p>
<p>Revenue from subscriptions and advertising was dependable and sufficient.</p>	<p>Subscriptions and advertising continue to be revenue sources but Mago now competes with a world of free content and ubiquitous information. The revenue processes are transmuting so fast, nobody knows what a sustainable and sufficient future for Mago looks like.</p>

Then

Staff roles were coveted and long held. Everyone worked in the office.

Teams, even technology teams, were stable with little turnover.

Now

Staff roles increasingly manage relationships with contractors, freelancers, and vendor teams.

The organization has staff all over the world, with only a core group that works full-time in the same office. People stay for 2-5 years on average.

Information was organized on pages.

Information could take any length, form or in digital relationship with other information.

Table 9-2. Technology patterns

Then	Now
Enterprise software with batch export scripts to share information to other enterprise software when necessary. Data is stored in multiple data stores.	A system of software reacting to asynchronous events. Increasing desire to stop copying data from one place to another and have a shared source of truth.
Relationships between information parts are structured by a single, hierarchical taxonomy.	Relationships between information parts are dynamic and evolving.
One technology team. No such thing as a systems architect.	Many teams and multiple systems architects.
Analog tools intertwined with digital tools in the content generation and delivery process.	Fully digital process with automation when possible.
No test coverage.	Layers of test coverage.

Then

Weekly releases.

Now

Many siloed release
processes.

Table 9-3. Process patterns

Then	Now
The sole focus is on delivering the magazine itself, supported by technology.	The focus is on delivering technology initiatives to support content and reader engagement.
Weekly planning meetings for both content creators and technology teams. Agile teams delivering in bi-weekly sprints.	Ad hoc team creation and delivery processes with little integration between them.
Hands-on production of materials.	Hands-on migration of content and assets across the digital ecosystem.
Budget supports better content.	Budget supports better technology.
“The way we’ve always done it” still works.	“The way we’ve always done it” is a blocker.
Ad sales are king.	Social media engagement is king.

Applying the Seven Questions to Mago

- **How does information flow?** Information that once flowed in a closed system, or one that could be understood as closed, now flows in an open system. It is consumed by many types of software as it flows.
- **What are the events** that happen in the system? Whenever content or assets are published, there are multiple destinations that shape and consume it. Users are engaging all over the ecosystem.
- **What are the boundaries** in the system? The boundaries in the system match the evolution of digital tools. The website, the app, the social media team. A rethinking of boundaries is critical.
- **What are the building blocks** in the system? Layers, information and storage are some of the building blocks across the system. Each layer has its own logic and components. The look and feel always needs to match the brand. The information needs to be structured for consumers and context. Storage needs to be fully secure. Some storage needs to be highly performant and some storage will be rarely accessed.
- **What is the delivery process?** It depends on what is being delivered.

- **How are people organized?** People are organized according to what type of technology product they produce. An app team, a web team, a video team etc.
- **How is discourse structured?** There is more politics than collaboration. Who has a voice totally depends on both charisma and the power structure they are in.

Now that we understand something about the patterns in the Mago system, where do we begin? What are the leverage points and how do we transform them? Generally speaking, they are in the flow of information and the core events. But there are, in fact, many ways to begin. As long as the recommendations are pattern aware, there isn't a right one ... only a right thing to try.

Summary

Pattern thinking is not simply learning patterns you can apply. It is the practice of discovering, discerning and describing patterns in your circumstances. There is not one right way to discover, communicate, design and discern patterns. But there are some relationships you can observe and questions you can ask to help illuminate them.

Counterintuitiveness can keep you stuck in old patterns while trying to form new ones. To discover the benefits of a pattern change, you need to try it and see. In systems, relationships produce effect and the patterns in those relationship interdepend. The same event can arise from very different patterns. It helps to understand, as best you can, the structures reinforcing those patterns before you act.

Patterns exist in relationship to time, events, context and structures. Patterns exist in your own mind. In most circumstances, external, technological and process patterns are impacting the software system. Nowadays, it matters to understand this because as software and information become ubiquitous, many organizations, like our fictitious Mago, are facing paradigm-shifting pattern changes.

Practices

Read:

- Donella Meadow's article, [Dancing With Systems](#)
- *Software Architecture Patterns* by Mark Richards
- *A Pattern Language* by Christopher Alexander

- *Design Unbound: Designing for Emergence in a White Water World* by Ann M. Pendleton-Julian and John Seely Brown
- *Design Patterns: Elements of Reusable Object-Oriented Software* by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.

Try to model an event happening in your circumstances. See if you can include the patterns and structures influencing the event. Don't worry if it's messy (or done by hand). The goal is to think through how relationships are producing effect.

Questions for Your Journal

- Consider an event happening in your circumstances. Explore it the way I've explored Mago. Use the types of relationships and seven questions to write about what patterns are involved. What are the external, technology systems and process patterns involved?
- What are the most impactful patterns in your circumstance? How are they causing the challenges you experience? What structures keep them in place? How are they supported by mental models?

¹ <https://thesystemsthinker.com/systems-thinking-what-why-when-where-and-how>

- | <https://learning.oreilly.com/library/view/enterprise-integration-patterns/0321200683/>
- | <https://donellameadows.org/dancing-with-systems/>
- | Modeling software to match the domain using input from domain experts
- | An approach to mapping business strategy and value streams

About the Author

Diana Montalion has 17+ years experience delivering transformative initiatives, independently or as part of a professional services group, to clients including Stanford, The Gates Foundation and Teach For All. She has served as principal architect for The Economist and The Wikimedia Foundation. She founded Mentrif Group, a consultancy providing technology architecture, systems leadership and workshops on nonlinear approaches. Writing, teaching and thinking about thinking are her favorite hobbies.