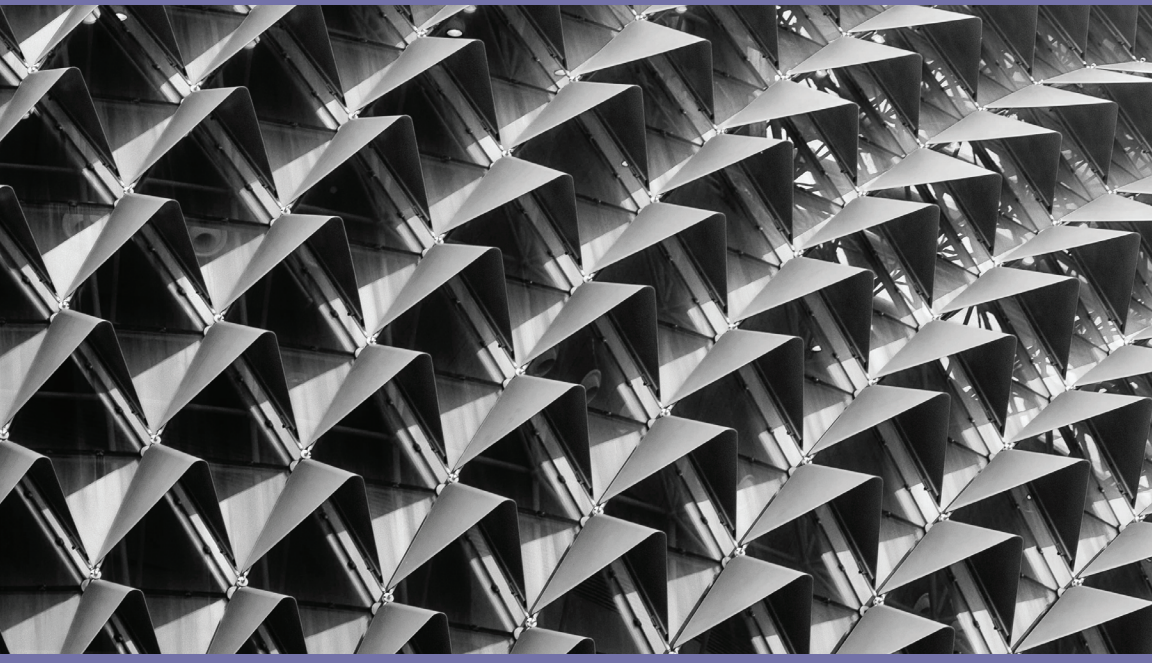
O'REILLY®

Understanding Design Thinking, Lean, and Agile



Jonny Schneider

Understanding Design Thinking, Lean, and Agile

Jonny Schneider

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Understanding Design Thinking, Lean, and Agile
by Jonny Schneider

Printed in the United States of America.

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

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

Editor: Angela Rufino
Production Editor: Kristen Brown
Copyeditor: Octal Publishing, Inc.
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Sara Michelazzo
Cover Photo: Jonny Schneider

July 2017: First Edition

Revision History for the First Edition
2017-07-13: First Release

978-1-491-98047-7

[LSI]

Table of Contents

CHAPTER 1
Introduction

Despite its title, this report is really about *ability*, *learning*, and *adapting*. Design Thinking, Lean, and Agile are mindsets that, when practiced, help organizations develop new competencies. We learn to tackle problems and explore possibilities. We strive to make every action a learning opportunity for making better decisions. And, we put learning to work as we pursue outcomes in a way that's optimized for adapting to constant change. More than following steps, procedures, or instructions, this report describes the mindsets and ways of working that help teams to think differently, practice new skills, and develop new ability.

Popular culture depicts designers as precious snowflakes. In the movies, developers are socially inept propeller heads. And we all like to joke about bean-counting middle managers and executives who are asleep at the wheel. And, all of them are petrified of being disrupted by Silicon Valley's hoodie-wearing startups.

Convention is dead in the modern age—there are no rules, and job roles are so last century. It's no wonder people are so confused about how to do software better.

These are exaggerated generalizations, I know, but they do paint a picture of the mess we face when working together to build stuff that matters. Creating digital products and services is a pursuit that requires collaboration across many disciplines. Technology, design, and business all have their own kinds of architects. Strategy has different flavors, from corporate to customer to technology. Products

require design, software is engineered, and someone needs to run the entire operation.

Design Thinking, Lean, and Agile are prominent *mindsets* among teams today. Each mindset brings its own kind of value to the product development life cycle (see Figure 1-1). And although they come from different origins—industrial design, manufacturing, and software development—they share many similarities, and are complementary and compatible with one another.

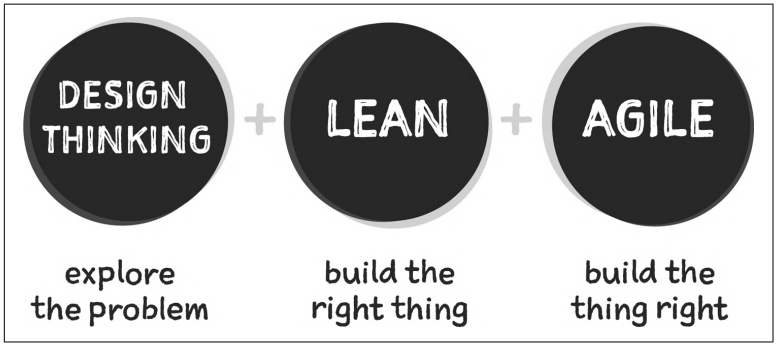


Figure 1-1. Design Thinking, Lean, and Agile

At a distance, Design Thinking is a mindset for exploring complex problems or finding opportunities in a world full of uncertainty. It's a search for meaning, usually focusing on human needs and experience. Using intuitive and abductive reasoning, Design Thinking explores and questions *what is*, and then imagines *what could be* with innovative and inventive future solutions.

The Lean mindset is a management philosophy that embraces scientific thinking to explore how *right* our beliefs and assumptions are while improving a system. Lean practitioners use the deliberate practice of testing their hypotheses through action, observing what actually happens, and making adjustments based on the differences observed. It's how organizations set their course, learn by doing, and decide what to do next on their journey to achieve outcomes.

The heart of Agile is building great software solutions that adapt gracefully to changing needs. Agile begins with a problem—not a requirement—and delivers an elegant solution. The Agile mindset acknowledges that the right solution today might not be the right solution tomorrow. It's rapid, iterative, easily adapted, and focused on quality through continuous improvement.

Although the strengths of each mindset come to bear more so in some areas than others, no single mindset claims exclusivity over any particular activity. Too often, people ask, Lean or Agile or Design Thinking? The answer is "and," not "or."

In this chapter, we take a detailed look at the origins of each mindset, their strengths, and how they all fit together. Then, in the following chapters, we explore how to bring it all together in practice to define actionable strategies, act to learn, lead teams to win, and deliver software solutions.

What Design Thinking Is

Popular opinion suggests that Design Thinking is all about squiggly lines, honeycomb diagrams, overlapping circles, and loop diagrams. See for yourself. Those visual models and processes are helpful for describing what Design Thinking sometimes looks like, but they don't really help anyone think practically or do things differently. And a cursory glance can leave the impression that it's just process or procedure. It isn't.

Design Thinking is a mindset as well as a toolkit of techniques for applying a designer's ways of thinking and doing. We can apply it in any context, domain, or problem. Design Thinking helps explore new territory and define a range of potential solutions. Design is a verb as well as a noun. It's something people do, not just a final result. It's a journey and a way of thinking as much as it is a final outcome.

Elements of Design Thinking

Design Thinking is about design as a *verb*. It's about the act of designing. Donald Norman, author of *The Design of Everyday Things* and the Godfather of user experience,[1] describes it nicely:

1 It's generally accepted that Norman created the term *User Experience Architect* when he joined the team at Apple in the 1990s. He says he created it because he felt that human interface and usability were too narrow: "I wanted to cover all aspects of the person's experience with a system, including industrial design, graphics, the interface, the physical interaction, and the manual."

> Designers don't search for a solution until they have determined the real problem, and even then, instead of solving that problem, they stop to consider a wide range of potential solutions. Only then will they converge upon their proposal. This process is called Design Thinking.

Let's look at the component parts in Norman's definition:

- Determining the real problem
- Searching for solutions
- Considering many options
- Converging on a proposal

He describes a discontentment with settling on the first solution. Ask yourself, when was the last time that your first idea was your best idea? Usually, first thoughts are the beginning—not the end—of finding the right solution. So much of design is about asking better questions, thorough consideration, and searching for possible solutions. The deeper our understanding of design's formalities (constraint, requirement, environment) are, the more avenues for exploration we have. The more we explore, the more potential solutions emerge. This creates choices, and converging on a final proposal is where the designer makes choices.

Divergence, Emergence, and Convergence

An important tenet in Design Thinking is intentional and repeated divergence and convergence. The British Design Council first expressed this in 2008 as part of a global study into how 11 leading companies apply design practice. Their Double Diamond, which is now ubiquitous as a simple visual model that demystifies some of the complexities of design practice, represents diverging (out) and converging (in) as the faces of two adjacent diamond shapes (see Figure 1-2).

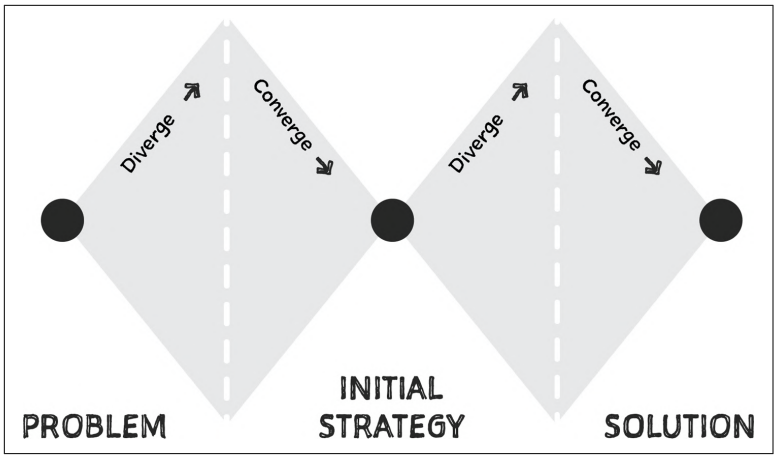


Figure 1-2. The divergence and convergence of Design Thinking

Stanford Design School, IBM, and IDEO also have well-recognized models for Design Thinking, wherein the concept is not represented so literally; rather, it's implied in the approach described by their respective models. These companies are credited with popularizing Design Thinking, but they also recognize that design is not only about process, it's about ability. Carissa Carter, director of teaching and learning at Stanford Design School, writes beautifully about the abilities that make designers great. Abilities like dealing with ambiguity, empathetic learning, synthesis, and experimentation. Process and tools are guiderails that help us to practice our abilities, but it's by doing design that we become better designers, not by following a process. For a fuller history on the origins of Design Thinking (hint, it didn't start with IDEO in the 90's) read Stefanie De Russo on Design Thinking.

Emergence is what happens when we practice design. The mechanics of emergence is pretty simple. Dave Gray, coauthor of *Gamestorming*[2] explains it best: "You will never find anything, unless you're looking for something."

Explorers of old understood this. Christopher Columbus set out westward from Andalusia intending to reach the East Indies to

2 David Gray, Sunni Brown, and James Macanufo, *Gamestorming: A Playbook for Innovators, Rulebreakers, and Changemakers* (Sebastopol, California: O'Reilly, 2010).

establish a new trade route. Surprise! The New World interrupted his journey halfway across the Atlantic Ocean.

Often, we don't find exactly what we set out for, but something better is found along the way. Being unsure of exactly where you're going or precisely how you'll get there *can* be a good thing. Design Thinking is about finding your way, just starting somewhere and keeping an open mind, but not an empty head.

Design Thinking Is for All

Design Thinking is not something special, that only designers do. Everyone designs, whether it's conscious or not. Design is not just the result of a designer working. It's an activity that is inclusive and collaborative. With a little thought, anyone can use the design mindset. This is increasingly true in domains not considered the heartland of design, like corporations, government, health, not-for-profit organizations, and education.

AirBnB was a failing startup in 2009. Challenging its beliefs about how to win, putting itself in its customer's shoes, and exploring solutions that don't scale is how AirBnB learned its way to outrageous success. Design Thinking firm IDEO partnered with Kraft, first to spark a change in how the company collaborates internally to innovate its supply-chain process design. Then, with some of Kraft's biggest customers, like Safeway, to transform operations for mutual benefit. And, finally scaling Design Thinking as a core capability at Kraft, enabling "joint value creation" with many more of their customers globally. Then, there's the Mayo Clinic Center for Innovation, where Design Thinking is at the heart of transforming the healthcare patient experience.

The popularization of Design Thinking over the past decade is changing what it means to be a designer. And it means more than ever that everyone participates in design.

What Lean Thinking Is

Lean is a management philosophy for improving any system that produces value—that is, any organization. Persistent improvement, high quality, and reduction in waste are some common characteristics of Lean management. Although these are some good outcomes, that's not what Lean thinking actually is. *Lean* is how Jones and

Womack described the ideas and behaviors they observed at Toyota's automotive manufacturing operations in Japan after World War II.[3] Behaviors like continuously improving manufacturing quality and efficiency through deliberate reflection and learning, and organizing work according to customer demand. Toyota's management practices—known as the *Toyota Production System* created by Taiichi Ōno[4]—were leading the company to outperform its western competitors. Table 1-1 summarizes these and other central ideas of Lean thinking.

Being Lean isn't achieved by following Toyota's recipe. That's because it's not procedural, it's cultural, and it requires change. Not just a change in the way work happens, but a change in the principles and values that motivate how people work.

This sounds familiar, doesn't it? We've all attended the company *all hands* meeting. Somebody high up speaks passionately about the need to change. We *must* improve and be more efficient. We *must* pursue quality in everything we do. We *must* be better. Workers file out of the auditorium, gossiping about whether the company is in financial trouble. Some speculate about a management shake-up or restructure. More seasoned workers have seen this before, remarking, "This happens every two years, a few months ahead of shareholder results announcements." Just days later, buried by the rhythm and cadence of business as usual, most have forgotten the speech that was supposed to motivate change throughout the company.

That's not how change happens. It doesn't work because it asks people to change, without changing the system and culture that exists around them. It's *how* we work, not just the resulting outcomes, that define Lean management.

From Scientific Management to Lean Management

Management practices forged in the smelters of the industrial revolution are still popular today. Then, mechanization and automation were transforming what it meant to work. Scientific management sought efficiency through process, rules, and control. Work was

3 James P. Womack, Daniel T. Jones, and Daniel Roos. *The Machine That Changed the World* (Free Press, 2007).

4 Taiichi Ōno, *Toyota Production System: Beyond Large-Scale Production*. (Cambridge, Massachusetts: Productivity Press, 1988).

standardized; disassembled into small, repeatable tasks; and then reassembled into a schedule of tightly managed procedures. If you've worked to a Gantt Chart that specifies the breakdown, dependencies, and timing of your work, you've participated in a style of scientific management. This reflects its core value of *control*.

This kind of management improved efficiency, but workers ultimately detested it, which led to stronger than normal trade unionism, mostly because work became de-skilled, meaningless, and demotivating.[5]

In modern software businesses, *control* through scientific management is a falsehood. Things are too complex, too unpredictable, and too dynamic to be controlled. Managers seek certainty and predictability where in fact there is little, and their management practices provide only a comforting blanket of untruth. Only when the project fails—often when it's too late to recover gracefully—does the lie become clear. Modern business calls for more modern management practices. In Chapter 2, we look at complexity and better ways to manage uncertainty.

Values and Principles of Lean Thinking

Lean thinking has been adopted and applied in many domains: healthcare, supply-chain planning, government, and education, to name but a few. Table 1-1 provides a snapshot of core values with related principles for the Lean management of anything.

Table 1-1. Values and principles of Lean thinking

Values	Principles
Learning and adapting over analysis and prediction	Test beliefs through doing, not analysis or planning
	Delay decision making to the last responsible moment
	Scientific thinking with deliberate practice
Empowered people are happier and achieve better outcomes	Define clear goals, trust teams and give autonomy to achieve outcomes
	Decentralize decision making

5 R. F. Hoxie. (1916). "Why Organized Labor Opposes Scientific Management," *The Quarterly Journal of Economics,* 31(1): 62, doi:10.2307/1885989.

Values	Principles
Outcomes over outputs	Performance is measured by whether value is delivered, not by how much work is completed
	Specify value, measure mostly that
Manage flow to optimize value	Reduce *batch size*
	Manage queues
	Deliver at speed
	Eliminate waste
	Respond to customer demand over creating inventory (create *pull*)
Quality is a result, not an activity	Build quality in
	Continuously learn and respond to improve things
	Pursue perfection

There are loads of helpful methods and techniques associated with Lean thinking, such as value stream mapping,[6] but simply implementing those procedures misses the point. We say Design Thinking isn't only about process and tools, it's about ability and practice. Lean is the same. If we take nothing else from Lean thinking, let us take Mike Rother's brilliant teaching of scientific thinking with deliberate practice, using *The Improvement Kata Model*.

The Japanese word *Kata* means something like "a set combination of movements performed as an exercise." The Improvement Kata (Figure 1-3) describes how we can practice the movements of scientific thinking. When we combine it with the plan-do-check-act (PDCA) cycle (Figure 1-4), we have both a way of describing the overall journey as well as some clear steps to help design and conduct the right experiments—to navigate from where we are (current condition) toward where we want to be (target condition).

But we also need *deliberate practice* to develop that ability. It takes practice because it's different from our default mode of thinking. Like many learned abilities—music, sports, arts—improvement happens faster when corrective input can be provided by someone who has already mastered the craft. So, there's a separate set of movements, The Coaching Kata, aimed at helping coaches teach.

6 Mike Rother, John Shook, and Lean Enterprise Institute, *Learning to See: Value Stream Mapping to Add Value and Eliminate MUDA* (Cambridge, Massachusetts: Lean Enterprise Institute, 1998).

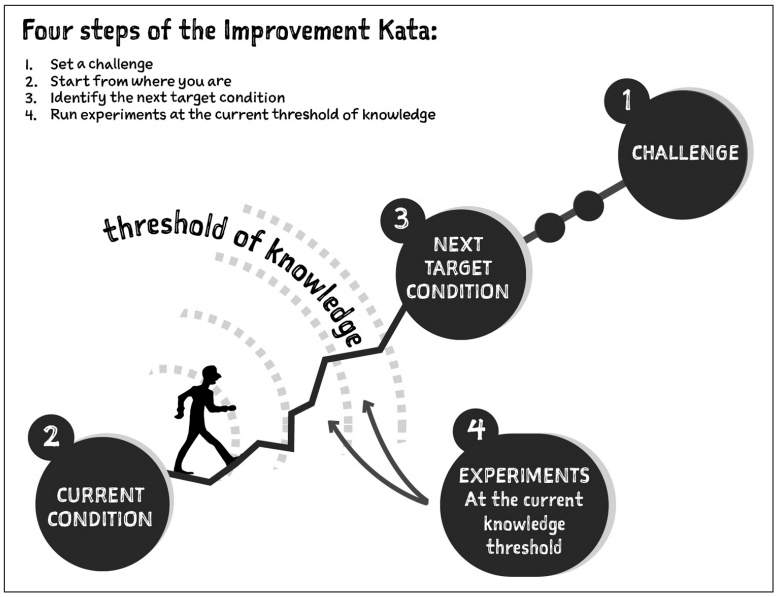


Figure 1-3. The Improvement Kata by Mike Rother (Source: The Toyota Kata Practice Guide (McGraw Hill, forthcoming in 2017)

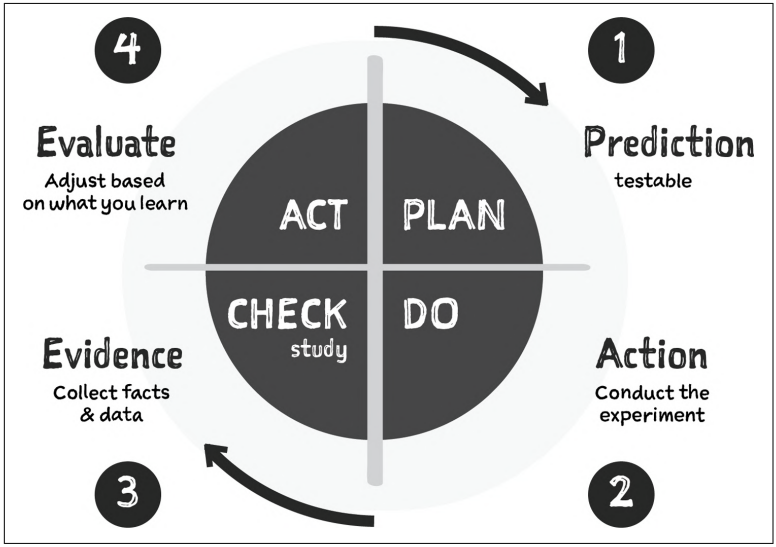


Figure 1-4. Interpretation of the PDCA loop by Mike Rother (Source: The Toyota Kata Practice Guide)

Following are the key concepts of the Improvement Kata:

- There's a *knowledge threshold*. That is, we don't know everything, and, often, how we think we'll get somewhere is not actually how we'll get there.
- By using the PDCA cycle to make predictions (*plan*), taking action (*do* an experiment), observing what happens (*check* the result), and interpret the meaning of evidence (take next *action*), we can systematically increase our knowledge, and iteratively learn our way toward the target condition.
- Deliberate practice is how we acquire and improve the skill of scientific thinking. That is, we must practice to learn, because scientific thinking is not our natural way.
- Being coached by someone who has mastered the craft provides corrective input so that we learn faster.

Too often, the conversation about Lean stops with the elimination of waste and a focus on quality in the context of process optimization. Those characteristics are undeniably important, but that's only part of what Lean has to offer. Lean is also fundamentally about learning, exploring uncertainty, making better decisions, and leading people to achieve outcomes.

What Agile Is

Agile came from a need to deliver software projects better. Like most movements, the true origins of Agile are debated. *Agile* as a label came from a collective of 17 independent software practitioners who coalesced in 2001 at a ski resort in Snowbird, Utah. That meeting resulted in the publication of the *Agile Manifesto*, and the formation of Agile Alliance. Some of the proponents of Agile at that time—Ken Schwaber who co-created Scrum, and Alistair Cockburn—were aware of the PDCA cycle and were somewhat influenced by the Lean movement. Certainly, there are similarities in the values and principles, making Agile and Lean thinking very compatible.

The Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work, we have come to value:

> **Individuals and interactions** over processes and tools
> **Working software** over comprehensive documentation
> **Customer collaboration** over contract negotiation
> **Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

At that time, software projects were predominantly managed using a heavyweight, so-called *waterfall* management practice. This describes the cascading nature of how work happens. Project phases like analysis, requirements, design, development, testing, and deployment run in a linear and consecutive sequence, where the preceding phases must be completed before continuing to the next phase. It wasn't working. Depending on whether you trust Gartner or Dr. Dobbs, somewhere in the range of 20 to 50 percent of software projects were failing.

Agile became the umbrella term encompassing alternative approaches emerging through the late 1990s for managing software delivery more appropriately. The purpose was not to codify the *right way*, but instead to describe the values, principles, and behaviors of teams that were embracing Agile ways of working and winning. Principles such as "welcome changing requirements, even late in development," "Working software is the principle measure of progress," and "Continuous attention to technical excellence and good design enhances agility" are some of the twelve principles describing the Agile way. Because Agile is *related* to Lean, it's unsurprising that the two mindsets share much in common. And mostly, they're differences come down to what they're applied to. Let's look at these similarities and differences in more detail.

Commonalities of Lean and Agile

Agile and Lean are similar in the following ways:

- They embrace and adapt to change, regardless of how late it occurs.
- They produce value iteratively, in short cycles.
- Each is humanistic, that is, valuing people above process, and encouraging autonomy and collaboration.
- Both Agile and Lean focus on quality, which in turn improves efficiency.[7]
- They seek to eliminate wasted effort.
- They continuously improve through reflection and learning.

How Lean and Agile Are Different

Now let's look at some differences.

Agile optimizes software delivery; Lean optimizes systems of work

Agile focuses more on *creating software*, whereas Lean is mostly about optimizing *systems of work* that produce value.

This is a little nuanced. It doesn't mean that Agile teams don't deliver value. Of course, they do—software has value. The point is that Agile was born out of a need for better ways to *deliver* software, and teams' primary measures of success are things like delivering working software, and the ability to adapt to changing requirements. Lean goes beyond software, addressing the entire value stream in a given organization. This *system of work* is a superset to software delivery. That is, software delivery is one activity (among myriad other activities) that organizations do to produce value for their customers.

7 Although that sounds opposed to conventional wisdom, it's true, and a fundamental principle of how Lean improves systems of work, and Agile delivers better software for less total effort. Episode 403 of *This American Life* explains how higher quality is achieved for less cost using Lean management at NUMMI's Fremont manufacturing plant in 1984, a partnership between GM and Toyota.

Continuous flow versus time-based iterations

Lean strives to achieve *flow* of value by aligning all work with customer demand. Work is selected according to customer demand and is *pulled* through the system that generates customer value. Lean principles—like managing queues, limiting work in progress, and reducing batch sizes—help to make the system of work efficient and optimize the work moving through the value stream. This flow is continuous.

Customer value matters in Agile, too, but the work is done in time-based iterations, whereby candidates are selected and prioritized according their value, size, and do-ability, from a giant backlog of possibilities. It's not that Agile can't be continuous and aligned to customer demand (we explore this in Chapter 5), it's just that effort is measured in iterations, and we use things like estimated team capacity, prioritization, and team velocity (actual throughput) to manage iterative completion of work.

Stable and repetitive versus always changing

As in manufacturing, a lot of Lean practice is about consistently producing the same output, over and over again, improving and optimizing each time. This is most obvious in the production of tangible things. Many Lean principles are also relevant in portfolio management, for which it's *initiatives* that are being *produced*. Although each initiative will have a dynamic outcome, the way we go about managing the portfolio of work is relatively stable, repetitive, and somewhat predictable.

It's the *soft* in software that means things don't need to be perfect and final on completion, never to be changed again. In fact, it's the exact opposite. The malleability of software lends itself to continuous change. Agile is all about responding to the constant need to change, and many of its practices—like Continuous Delivery (CD)—exist to do exactly that.

All Together Now

Design Thinking, Lean, and Agile mindsets aren't mutually exclusive. In fact, there's quite a lot of overlap. This is confusing, first because we often prefer simple explanations, and second because in the messy world in which we live, we tend to blend mindsets into ways of working that make sense for the job at hand. Some might

disagree, but this is for the greater good. There is nothing to be gained from dogmatic adherence to a particular *right way* to do things. On the contrary, when we thinkingly blend and combine different approaches in meaningful ways, we're exercising our innate ability as humans to solve problems. So often the question is "Lean *or* Agile?" The answer is really "and". It's Lean *and* Agile *and* Design Thinking. Some of the characteristics of each mindset are shown in Figure 1-5, helping to address a range of different needs.

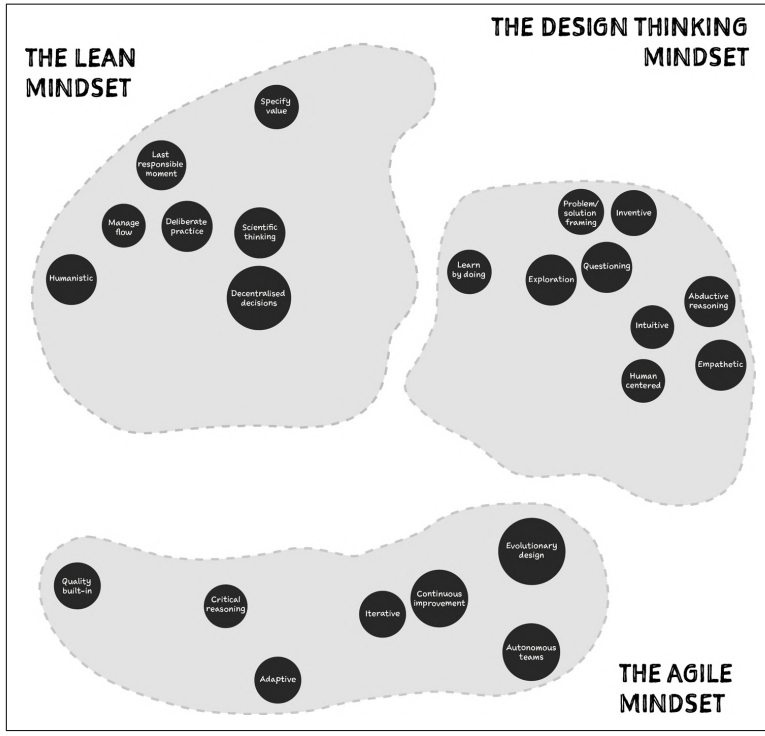


Figure 1-5. The characteristics of three mindsets

The Lean mindset drives continuous experimentation to learn our way to the correct answers. It helps in identifying the appropriate things to build as well as improving the system of work that delivers value. This is entirely agnostic to the medium in which value is produced; that is, it could be software, underpants, or healthcare.

The Design Thinking mindset is all about understanding constraints, seeing opportunity and exploring possibilities. It's a quest

toward finding opportunities and exploring solutions that create value for customers or the organization.

The Agile mindset is about achieving outcomes with software in the best way. It's how IT teams unlock value continuously, adapt to changing needs, and build quality into the software they create.

It's at the intersection of these three mindsets, depicted in Figure 1-6, that we see how everything can fit together.

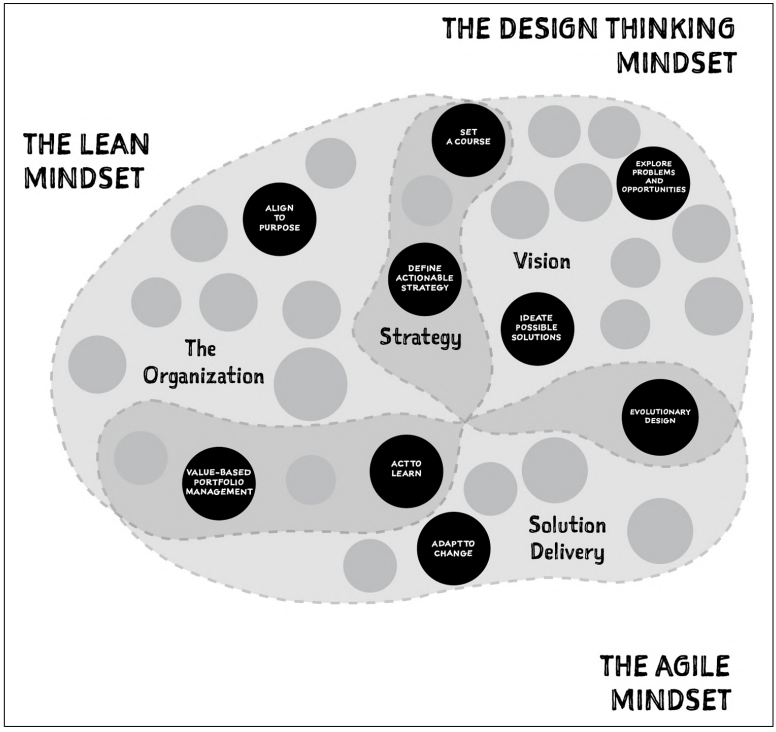


Figure 1-6. How the three mindsets work together

Together, Lean and Design Thinking helps us to understand where we're at today, where we want to be tomorrow, and pursue success through exploration, experimentation, and validated learning. The discipline of framing problems and opportunities and exploring many options in Design Thinking melds beautifully with the Lean practice of scientific thinking and learning by doing.

Design Thinking and Agile are a collaboration in realistic solutions. Software is the medium; engineers and designers are the artisans.

Together they craft solutions that deliver on desired outcomes. And they do their work iteratively, continuously, and paired together.

Agile and Lean is where strategy meets execution. Lean gives a framework for testing our beliefs and refining strategy through learning. This learn-by-doing approach works only if every part of the system is highly adaptive. Agile provides the flexibility to respond to change, which is a first-class capability for aligning technology delivery to real value, always.

The strengths of each mindset come together to help us achieve the right outcomes. Design Thinking is about *exploring problems and opportunities*, Lean moves us toward *building the right things*, and Agile is a way of *building things right*. Chapter 2 maps four steps for defining strategy and executing toward successful outcomes, highlighting how each mindset contributes along the way.

CHAPTER 2

Actionable Strategy

This chapter explores some of the challenges of complexity and the importance of learning and adapting. A four-step model describes how Design Thinking, Lean, and Agile mindsets come together and how you can apply them practically, along with a range of methods, tools, and resources to consider for getting the work done.

The Problem with Complexity

A big challenge for many products and services is that they're deployed into highly uncertain and complex systems, like our economy, a customer ecosystem, or even a large enterprise. These *complex adaptive systems* are made up of interconnected but autonomous entities, acting and reacting to one another, without centralized control. Predicting a specific result or outcome in these conditions is incredibly difficult because behavior in such a system is emergent. Although uncertainty is high, there are two characteristics we can be confident about in complex adaptive systems:

They're unknowable
: There are multiple truths, and the system cannot be described from only one perspective or language. That is, nobody really understands everything. And we can't fully know anything.

They're intractable
: That is, nobody is in control, the current condition emerges, and nothing is predictable.

This means that nobody knows what is going on, nobody can control it, and nobody can predict anything. Then there are humans—just one of the actors in the system—who are famously irrational in the way we behave.[1]

> There is no faster way of predicting the future, other than just going there.
>
> —Igor Nikolic

So, all strategies are full of risk and uncertainty. Those based only on modelling, analysis, insight, and perceived knowledge alone are more likely to fail than those based on action. It's not that those things are bad or pointless, but that action is a priority because it's by doing things that we learn most about what works.

Now let's consider the relationship between vision, strategy and action.

Vision and Strategy

There is an immutable tension between vision and strategy. The archetypical strategist is a deep thinker. A sage. A prophet. She seeks knowledge, analyses trends, and sees patterns. In pursuit of the goal, she plans action and anticipates reactions. Always observing, always ready, always ahead.

The archetypical visionary is a radical thinker. An idealist. A magician. She sees a future that defies today's logic. Contrary evidence is only a speedbump on the way to achieving unreasonable expectations. Unbending, unrelenting, unstoppable!

Vision sets direction, and strategy is how we get there. It's foolish to rush to action without some idea of where you're heading. Just as it's pointless to plan, but never do.

We must plan to act, act to learn, learn to win.

1 Dan Ariely, *Predictably Irrational* (New York: HarperCollins, 2009).

Morgan Raineri and Francisco Trindade started YourGrocer in 2013 in Melbourne, Australia. They offer fresh produce and groceries on-demand from local and independent stores, delivered same-day in the Melbourne area. It's a great service for time-poor metropolitans who want to support local and independent stores in their communities. Many Melburnians now enjoy this alternative to the duopoly of supermarket conglomerates that dominate the grocery market.

Early on, Raineri and Trindade's hypothesis was that *they were like their customers*: busy, professional, urban-dwellers who value fresh, good-quality produce, to fuel their healthy lifestyles. They also wanted to support their favorite local and independent businesses in their neighborhoods.

When interviewing some of their first frequent customers, some patterns began to emerge. They weren't who they thought they were. Their lives were different. Their early adopters were supporting *families*. Although these customers certainly cared to support independent businesses, a bigger motivation was to avoid the chaos of shopping with children in tow. Fresh and local produce is important to them, but more so for the welfare of their families, not just their own fussy preferences.

These customers are still motivated by *convenience*, but their needs, behaviors and problems are quite different than what Raineri and Trindade first imagined.

By recognizing this and adapting their strategy, as of this writing, YourGrocer has grown into one of Melbourne's most successful small businesses.

Defining Actionable Strategy

Strategy without action isn't a strategy at all. Yet, it's so common that goals and objectives are proclaimed, yet no strategic intent, overall approach, or basic guidelines are given on how to achieve them. There's some tension in the degree to which action should be specified. After all, top-down directives under *command and control* management often fail (because of the uncertainty inherent in complex systems) and are easily dismissed as rhetoric (not reflective of the day-to-day reality of getting the job done). On the other hand, fully decentralized, and uncoordinated action isn't always strategic.

If the left hand doesn't know what the right hand is doing, some activities might undermine others.

An actionable strategy is one that defines enough direction to allow action to occur. It's lightweight, rapid, and adaptable. Actionable strategy acknowledges that nothing is truly knowable, complex systems are intractable, and that learning and responding is the best path to desired outcomes. Figure 2-1 describes four steps for actionable strategy.

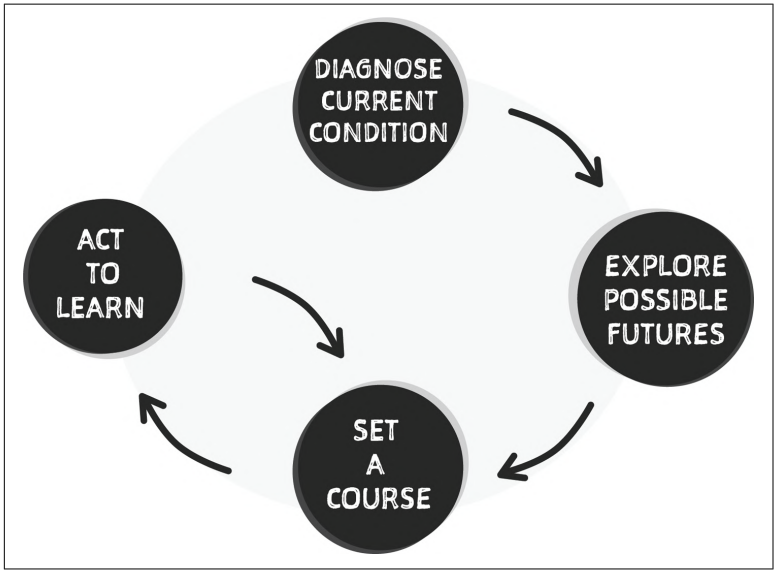


Figure 2-1. Four steps of actionable strategy

Step 1: Diagnose the Current Condition

A deep understanding of the current situation is the foundation for determining what to do about it. In his book *Good Strategy, Bad Strategy*,[2] author Richard Rumelt describes diagnosis as an exercise in thinking and imagination, and of judgment and evaluation. We gather information and determine the facts to identify problems and opportunities. We use critical analysis to make judgments about the meaning of what we know. We frame the problem, analyse information, and synthesis insights for strategic decision making.

2 Richard P. Rumelt, *Good Strategy Bad Strategy: The Difference and Why It Matters* (London: Profile, 2012).

> Diagnosis is a judgement about the meaning of facts.
>
> —Richard Rumelt

This first step draws on the focus and empathy for customer, intuitive reasoning and questioning that is baked into Design Thinking. This helps us to discover customer value and identify areas for further exploration. Lean brings critical reasoning and analytical judgment. This helps later when defining our first action and benchmarking our future success.

Table 2-1 details a range of assessment methods that are useful for understanding the situation, making judgments, identifying opportunities, and benchmarking.

Table 2-1. Approaches for diagnosing the current condition

Method	What is it	Further reading
Behavior mapping	A behavioral research method for observing the relationship between people and a place or environment. It reveals traffic patterns, interactions, and opportunities to optimize the service experience.	Doctor disruption on behavior mapping
PESTLE, SWOT, Porter's five forces	Structured business analysis techniques for exploring common dimensions of business.	Five forces, PESTLE, SWOT,
Five whys	Explore a problem space and find root causes by asking *why* five times.	Atlassian's guide to 5 Whys
Concept mapping	A sense-making method using words to visualize the complexities of a system by connecting many ideas and insights related to a given domain or focusing question. Concept maps help to elicit understanding and reveal new insight.	Novak and Cañas, Concept maps[a]
Ecosystem maps	Consider who's who and how value is created beyond your business in the overall net of who's playing.	Drawing your business ecosystem
Service ecology mapping	Think about the total set of needs of your customers and the ecology, not just the one's you're delivering upon.	Service ecology maps
Value stream mapping	See Table 3-1.	
Customer research	See Chapter 3.	

Method	What is it	Further reading
Service blueprints	Service blueprints visualize customer interactions over time (front-stage), while showing operations, technology systems, processes and staff interactions (back-stage) that underpin the service experience. A useful tool for diagnosing issues, and modelling future solutions.	ONE Design, Guide to Service Blueprinting Polaine & Løvlie, *Service Design: From Insight to Implementation*.[b]
KJ Analysis (aka affinity diagramming)	A fast, thematic analysis technique for groups that helps to identify areas of focus for further exploration.	Jared Spool on KJ analysis

[a] Joseph D. Novak and Alberto J. Cañas. (2010). "The theory underlying concept maps and how to construct and use them," *Práxis Educativa,* 5(1): 9–29.

[b] Andrew Polaine, Lavrans Løvlie, and Ben Reason, *Service Design: From Insight to Implementation* (Rosenfeld Media, 2013).

Step 2: Explore Possible Futures

By entertaining what might be, we generate options. Exploring possibilities is all about asking provoking questions, entertaining unconstrained thinking, following tangents, and new avenues of thought. We *create choices* before we *make choices*.

> Don't look for facts or answers—look for better questions. It's the questions we ask, and the meaning we explore that will generate the insights most useful to strategy.
>
> —Dr. Jason Fox

This is where we take advantage of the concept of emergence. By engaging in a quest to explore the possibilities, we discover new meaning, and it's that which often informs us where to go next. This is the homeland of Design Thinking. We've understood the problem or opportunity in step 1, now we're exploring many possible solutions, before later converging on our proposal in step 3.

Table 2-2 describes methods for exploring possible futures.

Table 2-2. Methods for exploring possibilities

Method	What is it	Further reading
Scenario planning	Get your futurist hat on and challenge yourself to think about multiple possible futures for your business.	Schwartz, *The Art of the Long View*[a]
Design charrettes (aka *crazy eights*)	A participatory design method where participants articulate many potential solutions to a given problem using fast sketches. Ideas are then compared and contrasted, with the strongest candidates selected for further exploration by the group.	Nielsen Norman Group on design charettes Google Ventures guide to crazy eights
Elito method	A generative design method using five building blocks to bridge the gap between analysis, synthesis, and potential solution designs.	Doctor disruption on Elito method
Prototyping	See Chapter 3.	
Design games	Collaborative innovation games designed to help teams overcome challenges and solve many different kinds of problems together.	Gray et al. *Gamestorming* (O'Reilly)
Business origami	An early stage design method for modelling the value exchange between actors in a system, based on selected scenarios. Great for rapidly exploring new possibilities within a system.	Jess McMullin on Business origami

[a] Peter Schwartz, *The Art of the Long View* (Currency Doubleday, 1996).

Step 3: Set a Course

This is where we *make choices* about which direction to take. We're evaluating our options, deciding what matters most, and homing in on strategic intent. Setting a course is more than simply declaring our goals and objectives. We must define an overall direction. Then, we need some guiding principles on how we believe we'll win, yet leaving out any specific instructions about precisely what to do.

> Be stubborn on the vision, but flexible on the details.
>
> —Jeff Bezos

This is about strategy and leadership. Lean thinking informs how we set challenges, and coordinate coherent action. Agile provides the means for creating technology solutions while keeping options open and remaining adaptive to change. The decentralization of control and leadership of autonomous teams is how we stay aligned to purpose and make better decisions.

Chapter 4 explores how we define our strategy, set direction and lead people on the journey to achieve the desired outcomes.

Step 4: Take Action and Adjust

Now it's time to test our beliefs through action.

> Vision without action is hallucination.
>
> —Thomas Edison

We make our hypotheses testable, run the experiments, measure the outcomes, and refine our initial strategy through learning. This is where strategy is quenched by action and made real. A strategy without action is merely speculation and conjecture. It's from action that we learn the most.

Chapter 3 includes a range of ways by which we can articulate our beliefs and test ideas. Chapter 5 describes how Agile software delivery enables organizations to adapt to change at scale.

Conclusion

The four steps to actionable strategy are like the glue between these mindsets. The fundamentals of Design Thinking are put to best use in the first two steps, where our intent is to understand today's reality and explore the possibilities for the future. The Lean mindset comes to bear in steps one, three, and four, where it's all about identifying the best opportunities to pursue, setting direction, and learning our way to success through deliberate experimentation. Much of this experimentation might not involve writing a line of code—after all, working software is still an experiment, just a really expensive one. As confidence increases, and software *is* the experiment, Agile is how teams constantly adapt to change, repeatedly adjusting their course and taking next steps (step 4).

CHAPTER 3
Act to Learn

This chapter is all about how to learn well, for better decision making. We explore how to articulate our beliefs and riskiest assumptions, and then design experiments that help us learn the relevant things. To help put theory into practice, there's loads of methods for validating problems, evaluating potential solutions, and testing market demand.

Strategy is about doing. Doing is how we learn. Learning is how we win.

Action is how we push a theory toward reality, and in our complex world, winning is often about how we learn and respond, not how much we *know*.[1]

Learning your way to a strategy that works begins by doing the following:

1. *Defining* your beliefs and assumptions (so that they can be tested)
2. *Deciding* the most important thing to learn, and how you'll learn it
3. *Designing* experiments that will deliver learning

1 Remember, nothing is truly knowable in complex adaptive systems.

Defining Your Beliefs and Assumptions

Before we talk with customers or run an experiment, we need to identify what we need to learn. Otherwise, we'll have a lovely chat, but might not learn anything that lets us know we're on the right track.

The *problem-assumption model* in Figure 3-1 helps break down our beliefs and identify the underlying assumptions in our thinking. Those assumptions are the basis of what we test. We can translate them into questions for customer interviews or use them to design experiments that create a measurable result.

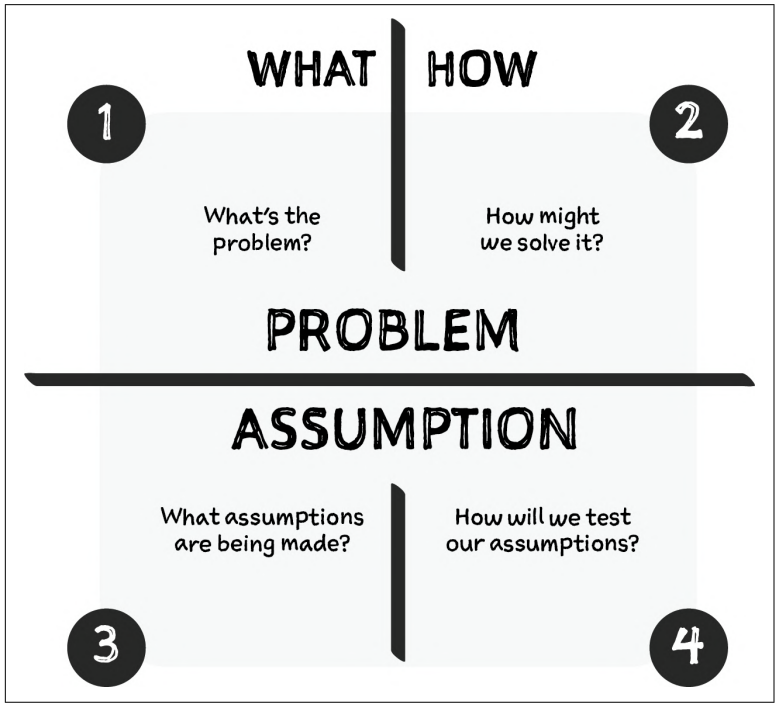


Figure 3-1. The problem-assumption model helps express problems, solutions, assumptions, and questions (source: created by Jonny Schneider and Barry O'Reilly)

Imagine you're a digital director at a personal investment company. Your competitors all have robo-advisory services,[2] and the executive leadership team feels they ought to have one, too. You have three months and $2 million to make it so.

The *solution* is a robo-advisory service. We think it solves the customer job of managing a personal investment portfolio. What *assumptions* have we made?

- Traditional advisory services are out of reach for a significant cohort of customers.
- People will trust automated advice enough to make financial decisions.
- They're willing to pay something for the service.

Now we can design questions and experiments to find out if that's true, and understand why or why not.[3]

The *problem-assumption model* is flexible. You can begin from anywhere—problems, solutions, assumptions or questions—and elaborate to fill-out your thinking. Many people begin with solutions and then explore the problem being solved, later moving on to the implied assumptions. As adoption of Design Thinking continues, more and more teams begin with the problem and then elaborate their solutions, assumptions, and questions.

Decide What to Learn and How to Learn It

Know what you need to learn. That sounds obvious, but it's surprising how often teams choose research methods that can't deliver the insight needed to move forward. Overuse of online surveys to quiz customers about their desire, intent, or behavior is one such antipattern. A well designed and executed survey has its place, but so much of the time, product teams will learn more through other methods

2 Robo-advisors are a class of financial adviser that provide mostly automated financial advice based on mathematical rules or algorithms.

3 In this case, we can infer that there is some truth in these assumptions, given that companies like Vanguard, Nutmeg, and WealthSimple have up to $50 billion in assets under management.

like observation or conversation. Another antipattern is conflating a good result in a prototype test with strong customer affinity with the problem or demand for the solution. Without being confident in the problem being solved and measuring the true demand for the solution, teams risk shipping awesome products that customers are unaware of, don't care about, or never use.

In 2015, a team built a mobile app that puts money back in the hands of travelling consumers by brokering the complicated transactions that occur between merchants, government agencies, and consumers for tax-exempt international retail shopping.

This brokering service was provided by a market leader with significant share of tax-free shopping refunds. The goal was to digitize the process to remove pain—and financial loss—for customers lining up in long queues at airports to submit complicated paperwork.

The team set out to understand the problem, and explore the right solutions. During solution design and prototyping, it believed that providing the right information about how the service works as part of a well-designed on-boarding experience was key to creating *stickiness* and activate customers to use the app while they shop.

They had a killer on-boarding experience, that didn't solve the real problem: the app was just one of 90% of all apps that are *zombies*.[4] These are applications that don't list in any top 300 list on Apple's App Store, in any location, for any category. That is, they can't be discovered organically, you'd need to search for the name to find it.

What couldn't be solved in solution design was how to reach customers in the first place and pique their interest with our promise to solve their problem. The team built an app with a better-than-average customer experience that solved real customer pain. Yet it failed. Not because it's a bad app. But because nobody knows about it, and nobody uses it.

4 "The App Store in 2016: zombies, lists and ranks," Adjust, *https://www.adjust.com/resources/app-zombies-2016/*.

Research and Experiments for Learning

Sometimes, we learn things that we weren't expecting to learn. Perhaps more often, we don't learn enough to make conclusive or definitive decisions. Finding the signal in the noise is challenging enough, so anything that reduces bad ambiguity[5] and helps us to look in the relevant places with the appropriate tools is a good thing.

Following, is a range of ways to think about learning, along with practical methods that help us to carry out the work:

- Validating the problem
- Evaluating potential solutions
- Testing market demand for a solution.

Problem Validation

Many solutions fail because they solve no meaningful problem. Charlie Guo learned that lesson the hard way. We fall in love with our ideas and our biases get the better of us. We focus too much on the solution, without properly understanding if there's really a problem worth solving.[6]

> I don't want to start another company until I find a problem that I care about. A problem that I eat, sleep and breathe. A problem worth solving.
>
> —Charlie Guo, cofounder of FanHero

Customer Problems

To understand customer value, we must know customers.

Go to where they are. Watch them. Talk to them. Build an understanding of what it's like to be them. Challenge our assumptions about what matters to them, how they behave, and what they need. This seems simple—and it is.

---

5 Good ambiguity is the kind we explore to find opportunities. Bad ambiguity is where our learning is questionable because we're not confident in our approach to learning.

6 That's not the only reason things fail. Sometimes the solution just isn't very good. Or, a competitor eats you for breakfast. Or you're too early or too late. Or, or, or...

> No facts exist inside the building, only opinions.
>
> —Steve Blank

Organizational problems

Sometimes the problem to solve is an organizational one, not a customer one. In this case, it's not a customer problem that we need to validate; rather, it's a problem or inefficiency in the way we're solving customers' problems.

> Listening in on a colleague taking a customer call at an internet company in 2015 provided an example of a typical organizational problem. The agent—let's call him Alan—spent more than an hour resolving a configuration issue to get a customer's internet service working again. The customer, Sarah, was thankful, and at the end of the call, Alan followed protocol, asking, "Is there something else I can help with?" Sarah did have a question about her most recent bill. First, Alan explained that he was going to call Sarah back in a moment, asking that she stay on the line and complete a short survey about the level of service she'd received so far. A few minutes later, Alan called Sarah back and then transferred her to billing and payments, where she sat waiting for a further 15 minutes before an agent became available to respond to her query.
>
> When asked about it later, Alan's rationale for doing these strange manuevers became clear. His *service score* was low that day, and he needed to make numbers. If he transferred Sarah directly, she might not have completed the survey, or might have given a low score for something out of Alan's control. Even though Alan had fixed the main problem for Sarah, if he then transferred the call, it wouldn't register as a *first call resolution,* an important performance metric for agents at the company. Also, by splitting it into two separate calls, his *average handling time* was effectively halved—another performance metric that was important to Alan that day.
>
> Even though these performance metrics are designed to improve customer experience, that's not what happens. Instead, customers are routinely bounced around from one department to another, which causes frustration from them, and increases the total cost to serve for the organization. Nobody wins.

Table 3-1 describes many options for identifying and validating meaningful problems.

Table 3-1. Learning for problem validation

Method	What is it	Why it's good	Note
In situ consumer study	Observation of behavior in the context that it occurs.	See how people interact with the product/service/ brand from their point of view.	
Design probes	Study of participants behavior and attitudes over a period of time, often recorded in a journal.	High-quality data, collected close to when events occur.	How to do a design probe
Qualitative interviews (semi-structured interviews)	Interviews with customers to explore their attitudes, needs and problems.	Fast way to test early assumptions, or discover real customer needs. Flexible.	Mia Northrop[a] on developing your interviewing technique
Surveys	A structured questionnaire to gather data on a specific research question.	Reach a large number of people cheaply and efficiently.	Designing a statistically accurate, unbiased survey is a skilled activity
Experience mapping	A visualization showing the outside-in view of the end-to-end experience for customers.	Pin-points problems and creates alignment of business goals to customer value.	Adaptive Path guide to experience mapping
Analytics	Quantitative analysis of measured behavior.	Data is empirical, representing actual behavior, not reported behavior.	Learn nothing about *why*
Value stream mapping	A process map showing the inside-out view of everything that happens in the organization to deliver value to customer.	Lightweight, paper and pencil study. Helps to understand time-to-completion and identify waste and bottlenecks in the process.	Rother & Shook, *Learning to See*[b]
Demand study	An empirical study of how value/failure demand flows through an organization from concept to cash.	Understand the volume of rework versus value generating work that is happening. Identifies inefficiencies, problems, and opportunities for improvement.	Vanguard on failure demand

[a] Mia is a terrific design researcher—the queen of qualitative interviewing!

[b] Mike Rother, John Shook, and Lean Enterprise Institute, *Learning to See: Value Stream Mapping to Add Value and Eliminate MUDA* (Cambridge, Massachusetts: Lean Enterprise Institute, 1998).

Solution Evaluation

Table 3-2 describes ways to evaluate how well our proposed solutions solve a given customer problem. It's the familiar ground of prototypes, analytics and testing with customers.

Table 3-2. Learning for solution evaluation

Method	What is it	Why it's good	Note
Concept prototype	Low-fidelity, throw-away sketches and mock-ups, for rapidly exploring concepts with customers	Fast, low cost. Participants are more comfortable to give critical feedback, because sketches are low effort.	
High-fidelity prototype	A detailed and interactive mock-up of the product experience	Validates the nuts and bolts of the solution like interaction design, content, look and feel. Easy to iterate and build upon based on feedback.	
Concierge	Personalized service provided to a small cohort of early customers, to learn what works before building an automated solution.	Generates solution options through exploring the problem with customers.	Difficult to scale. Risk building for a niche because cohort might not represent the market.
Working prototype	A limited implementation of a product, focusing on the *happy path*, and tested with a pilot cohort of customers to measure how well the solution performs.	High confidence in results because it's real software with real customers.	
Multivariate/ split tests	Testing multiple variants of a solution with cohorts of live customers to quantitatively learn which elements perform best.	Particularly effective for optimizing an existing product or service with high volume of traffic, and low cost to deploy working solutions as software.	

Demand Validation

Validating true customer demand is where we ask customers to put their money where their mouths are. Our goal is to measure behavior that indicates demand for the solution we are offering, in real market conditions.

It's the reverse of *build it, and they will come.* Demand validation says, *when they come, build it.* Measuring demand helps with choosing where to invest or whether to invest at all. Before spending money to build what we think people want, we aim to measure what solutions they actually want or need. Table 3-3 describes ways to validate demand.

Table 3-3. Learning for demand validation

Method	What is it	Why it's good	Note
Facade ads	A targeted search marketing ad directing customers to a simple landing page about the product.	Easy to target specific groups of people and experiment with different variants to see what resonates most.	
Preorders	Get paid customers signed-up to a product or service before it has been created.	Strong signal of demand, as customers put their money where their mouths are.	
Wizard of Oz prototype	The appearance and experience of a complete and working service, where all of the back-of-house processes are carried out manually.	No investment in supporting infrastructure required. A realistic test of actual market demand.	Zerocater started with a spreadsheet
Competitive market analysis (for existing markets)	Quantitative data and reports from independent agencies on market size, growth trends, geographies, segments, etc.	Find out what has or hasn't worked for companies competing in the same space.	Not very helpful for nascent markets!
Google trend analysis	Compare search trends using Google search data by time, location and other dimensions as a proxy for customer interest.	Free, fast, and kinda fun.	• Coconut yoghurt versus Buddha bowl • Triumph Thruxton versus Ducati Scrambler

The Cost of Experiments

We're looking for the shortest path to find out whether our strategy works. We want to move fast, challenge our beliefs, and mold the appropriate strategy, while also investing enough time and effort to properly explore uncertainty. Knowing when to stop exploring and commit to something in market is a balancing act.

Looking at experiments by *cost* and *confidence* helps guide us to select the best approaches for our circumstances (see Figure 3-2).

Cost

- Time
- Effort
- Resources
- Financial cost

Confidence

- Accuracy of data or insight
- Usefulness to inform decisions

A costly activity is one that takes time to prepare and execute, and involves significant people and resources. Traditional research and high-fidelity prototypes are examples of costly activities. A low-cost activity could be a sketch and a conversation examining a specific question you're exploring.

An example of an activity that produces accurate results—results in which we can have high confidence—might be a preorder campaign, for which prospective customers literally vote with their money, promising to pay if you deliver on the product. An activity with low accuracy results would be a competitor analysis of the current market.

Activities that are high-cost or low-accuracy are not *bad*. It's about choosing the best activities for a given stage of product development. It's about knowing what you need to learn and determining how you'll learn it.

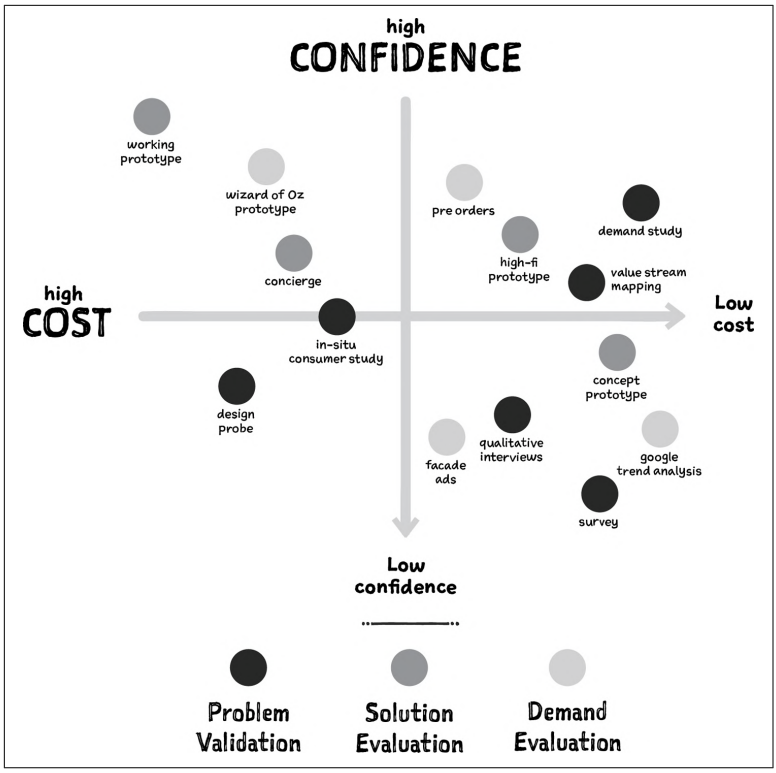


Figure 3-2. The cost of experiments

Conclusion

Product success relies on a lot of stars lining up properly. We need a problem worth solving, access to customers who need what we're offering, a solution that's technically feasible and commercially viable, a good sense of market timing, and a measure of good luck. Developing solutions is an economic activity, taking a lot of time, money, and effort. And, great products fail all the time, for a wide range reasons. Our goal here is to learn quickly and fail fast.

Design Thinking helps put the customer into focus and brings empathy to problem solving, along with creativity and innovation to solution exploration. Lean gives us a framework for scientific learning. We identify what to learn, and run experiments that help us make decisions as we navigate uncertainty. Although a great deal of learning can happen faster, cheaper, and more effectively without writing a line of code, Agile software development still has a big role

to play. The cost of creating software continues to decrease, meaning that many organizations are choosing to move to software-as-an-experiment earlier. Software prototypes with real customers can be a great way to learn what really works, especially when building new products. When it's an existing product/service, quantitative analytics and split/multivariate testing create feedback loops that help Agile product delivery teams decide what to do next.

CHAPTER 4
Leading Teams to Win

In this chapter, we look at ways to communicate vision and purpose, and how to align teams achieve success. We explore the mechanics of autonomy with the protocol of coherent action, and introduce some methods that help teams to make decisions and prioritize along the way.

Let's be honest: a lot of the work that goes on in modern businesses can seem void of meaning. Most of us aren't providing critical services, saving lives, or curing diseases. Mostly, we're building software products and services. And mostly, we're doing it for commercial gain. Maybe not our own individual gain, but at least the gain of our employers, or their shareholders, or capitalism at large. That's not very motivating. For those with a strong work ethic, hard work is a virtue, but the work in itself isn't always virtuous. The next generation of talent are set to challenge the status quo of *hard work for fair pay* further yet, and they're estimated to comprise 75 percent of the workforce by 2025. Organizations are desperate to find ways to make work more meaningful, purposeful, and engaging for people.

People want to be self-directed, trusted, and empowered to do the right thing. Even in seemingly mundane workplaces like the Faber-Castell pencil factory in Nuremburg,[1] people are happier, more engaged, and do right for the company when they're included and valued as people, not just *resources* or *units of effort*. A senior

1 To me, a pencil factory is actually fascinating, along with many other surprisingly awesome things.

manager at Faber-Castell chalks this up in part to the culture of empowerment at the company.[2]

> The more you give people a say, the more they help the company to win.
>
> —Senior manager, Faber-Castell

A highly engaged, purpose-driven, and empowered workforce. Great! But, unless everything is coordinated and aligned, efforts easily become incoherent and potentially counterproductive. Just like the following story of a motivated, passionate, and engaged team that ultimately failed because it's members cared about different things and were pulling in different directions.

It's easy to remember being part of a dysfunctional team. It doesn't happen because people are unkind, foolish or malicious.[3] Things become wonky when people have opposing motivation and are aligned to different goals.

Working in one team of high-performers building a new product, I felt a lot of tension between colleagues. It grew worse over time, and we became increasingly dysfunctional. I couldn't understand why at first. It was an exciting project. We had license to reimagine the product experience. We were empowered to choose the best technology stack and architecture. We were working on a problem worth solving. And, we had an "A" team.

Then, I heard a remark from a developer to his friend on the street at lunchtime: "How are we supposed to build this thing when there aren't any real requirements? A hypothesis isn't enough. This is unbelievable!" The tone of disbelief in his voice made it clear. Our team had become dysfunctional because we were motivated to achieve conflicting outcomes. Designers were exploring possible solutions while analysts sought certainty of requirements. Engineers were building robust and scalable software while strategists were chasing problem-solution fit.

2 Carl Deal, Tia Lessin, and Michael Moore, *Where to Invade Next*. Directed by Michael Moore (2015).

3 On the contrary, I've found that people are mostly intelligent, helpful, and kind.

We could have avoided a lot of turmoil and angst by acknowledging what stage of product development we were at, having clarity of purpose, and a shared understanding of the right goals and outcomes for our work.

As it turns out, that's not so easy!

Sometimes it feels as if getting people on the same track is the most difficult thing facing any team. Having a clear direction, getting everybody aligned, and taking coherent action is critical. Let's look at how to do that.

Purpose-Driven Autonomy

In *Drive,*[4] author Dan Pink makes a compelling case that incentives and bonuses are a bogus mechanism for getting people to perform at their best. In studies of human motivation, he finds that giving incentives leads to poorer performance for tasks that require even just a little cognitive ability.[5]

Instead he argues that autonomy, mastery, and purpose are key to motivation and engagement:

Autonomy
: The urge to direct our own lives.

Mastery
: The desire to get better and better at something that matters.

Purpose
: The yearning to do what we do, in the service of something larger than ourselves.

Mastery in large part is about passion. You can't necessarily make someone passionate about something. That's intrinsic. But we can do things to make work more purposeful, and create an environment in which teams and individuals have autonomy to do their best work.

4 Daniel H. Pink, *Drive: The Surprising Truth About What Motivates Us* (Penguin, 2011).

5 Incentives work just fine for predictable, repetitive, procedural activities. The catch is that most work in modern businesses requires problem solving and creativity. This work suffers under incentives.

We can communicate purpose in our vision. We can include everyone when defining desired outcomes. And, we can give teams autonomy and accountability to determine their own strategy.

Perhaps nowhere else is this so critical, as it is in military operations. Armed forces have formalized their guide to action as military doctrine. Let's look at what we can learn from how they run operations.

Mission Command

Warfare is unpredictable and dynamic. Information is incomplete and situations unfold quickly. Agility and adaptation are critical to success and survival.

The Army Doctrine on Mission Command (ADRP 6-0)[6] is fascinating and covers the principles of Mission Command in great detail. Fundamentally, Mission Command describes how armed forces do the following:

- Expect that plans will change
- Train personnel to be agile and adaptive in any situation through disciplined initiative
- Communicate clearly the intent of the mission
- Allow teams and individuals on the ground to pursue mission outcomes as they see fit, adjusting to the situation as it unfolds.

The ADRP says:

> *Mission command* is the exercise of *authority and direction* by the commander using mission orders to enable *disciplined initiative* within the *commander's intent* to empower *agile and adaptive leaders* in the conduct of [unified land] operations[7]

There's quite a lot in there. Let's unpack it.

Mission command is how the US Army maintains *centralised intent* and *dispersed execution* through *disciplined initiative*. It's how it coordinates many teams, performing complicated activities in dynamic environments, all aligned to common goals that help teams

6 US Army, United States Government, Army Doctrine Reference Publication Adrp 6-0 Mission Command (2011).

7 Formatting (italics and parentheses) is my own.

achieve certain outcomes. Importantly, this is done without dictating how the mission should be executed: that is left to the judgment of subordinates. Those who are closer to the action are empowered to decide the best course of action—within certain guidelines—to respond and adapt to changing circumstances in order to achieve the desired end state.

The commander's intent is a clear and concise expression of the purpose of the operation:

> Commanders provide subordinates with their intent, the purpose of the operation, the key tasks, the desired end state, and resources. Subordinates then exercise disciplined initiative to respond to unanticipated problems.

Disciplined initiative is what combat personnel are trained so rigorously to do during tactical training. This is their *deliberate practice*, preparing them to perform in uncertain and dynamic circumstances with initiative:

> *Disciplined initiative is action in the absence of orders, when existing orders no longer fit the situation, or when unforeseen opportunities or threats arise.*

All Together Now

We can coordinate people in teams, working on initiatives in a portfolio, using the protocol of Mission Command.

It's remarkable how well Mission Command aligns with the principles of Lean and Agile explored in Chapter 1 and listed here:

- Learning and adapting over analysis and prediction
- Responding to change over following a plan
- Empowered people are happier and achieve better outcomes
- Individuals and interactions over processes and tools
- Outcomes over outputs.

Mission Command expects the unexpected (threshold of knowledge), trains people in the skill of responding (deliberate practice), sets clear outcomes with some guiding principles (the Commander's intent), and decentralizes control, relying on people that are closest to the action to make the right decisions (autonomy).

In Chapter 2, we introduced four steps for defining actionable strategy, and explored how Design Thinking, Lean, and Agile contribute along the way. Chapter 3 showed us the skill of learning, both as a way to solve problems and find opportunities, but also to explore uncertainty. Now let's consider ways to visualize strategy, coordinate action, and make decisions along the way.

Techniques for Communicating Purpose and Progress

What follows are a range of techniques that help to articulate the purpose of missions and communicate team's progress in achieving outcomes.

Visual Management

Make anything visual, and you make it much more comprehensible. Better yet, when people collaborate and visualize things, they build a shared understanding together. Visualizations help us to articulate a common purpose and tell a convincing story to persuade others to join our quest. They describe *The Commander's Intent*. But far from being directive, visual communication is democratic, inclusive, and participatory. It draws people in, stimulates collaboration, and generates shared points of reference. Visualizations describe an understanding and anchor teams. Author Jim Kalbach calls them *alignment diagrams* and explores the topic in much more detail in his awesome book, *Mapping Experiences*.[8]

Table 4-1 lists some visualization methods for leading teams, and Figure 4-1 illustrates a *product design wall*, showing how action is connected with strategy for a team.

8 James Kalbach, *Mapping Experiences* (O'Reilly, 2016).

Table 4-1. Visual models for aligning work to purpose

Method	What	Why	Further reading
Vision boards, storyboard canvas, pitch walls	Collaboratively build the story of your strategy and aspirations with colleagues, and use the visual narrative as a way to communicate your mission to others	Build common purpose among teams. Refines your story and helps persuade others to support your quest.	
Product design wall	An information radiator showing the product vision, strategic intent, and progress toward the desired end state.	Combines strategy, design and engineering into one shared view that everyone can relate to.	See Figure 4-1
Strategy constellation diagrams	A visual model that shows the relationship between product initiatives in context to a broader program of work	Aligns many different initiatives-not just products-to a common purpose, and serves as a single view of strategy for all teams.	Jonny Schneider on constellation diagrams
Portfolio/ roadmap visualization	A visual system that show the process, the people and the work in one view for a given portfolio of initiatives	Single point of reference for all teams to know what work is in play, where it's up to, and who's working on it. Aligns disparate activities into coherent and coordinated action.	Mind the Product on portfolio roadmaps with gov.uk
Obeya rooms	The ultimate in visual management. Obeyas represent everything that is happening across all parts of a business into a single *wall of walls*, often consolidating the collective activity of tens or hundreds of teams in large organizations.	Acts as a *command center* for leaders of the organization. Breaks down departmental thinking, aligns all work to purpose and strategy, and makes issues and deviations easy to identify, follow up, and act upon in real time.	IndustryWeek on Obeya

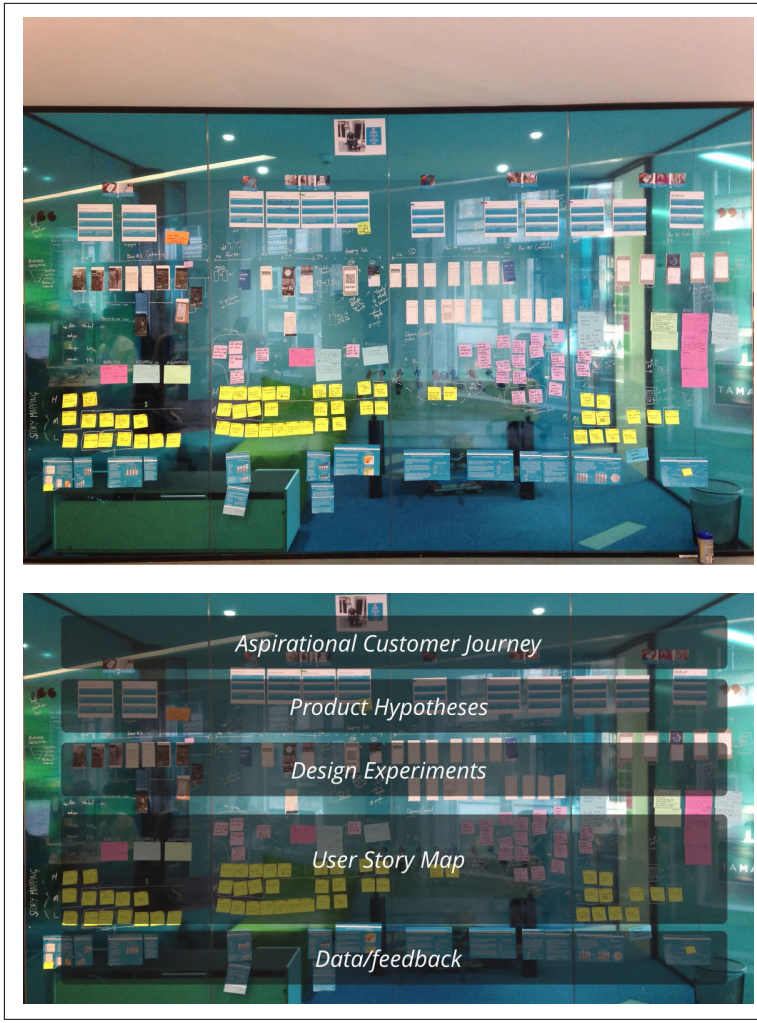


Figure 4-1. A product design wall

Techniques for Prioritizing Value

It's an awesome thing to have a shared vision, an initial strategy about how you're going to get there, and the right people and resources to make it happen. Teams now face three critical questions:

- What shall we do first?
- How will we know if it's working?
- What do we do next?

Simple as these questions sound, the answers aren't always obvious or straight forward. Especially when there are *many options* and *limited capacity* to do them. Those conditions seem true for every product team ever; this problem is universal. There aren't enough people—and never enough money—for organizations to do all the things they want. They must choose and decide. In our personal lives, we make decisions continuously about what our goals are and how we'll spend our time and resources in pursuit of them. At the park, a thirsty puppy chooses between a slurp of water (biological needs) and playing with his pals (social needs) during his morning walk.

But acting on instinct is dangerous. Buster Benson, John Moonigan, and others highlight all the ways we let our biases get in the way. The themes they describe here don't help us to make objective choices:

> *To get things done, we tend to complete things we've invested time and energy in.*
>
> *To stay focused, we favor the immediate, relatable thing in front of us.*
>
> *We simplify probabilities and numbers to make them easier to think about.*
>
> *We favor simple-looking options and complete information over complex, ambiguous options.*

Let's look at some ways to make better decisions.

Dimensions of Success

To know that you're being successful, you need to know what success looks like. The *dimensions of success* you define ought to have direct correlation with your vision and strategy. Ask specific questions. What ways might an activity contribute to a strategic outcome?

This kind of subjective reasoning can be useful for initial prioritization of what to do first. At this stage, our threshold of knowledge is often pretty low, mostly because meaningful action is yet to be taken. It's okay to use our intuition—and any empirical data that's

available—to make initial judgments about the potential of a set of options to deliver on the desired outcome. Figure 4-2 shows an example of this.

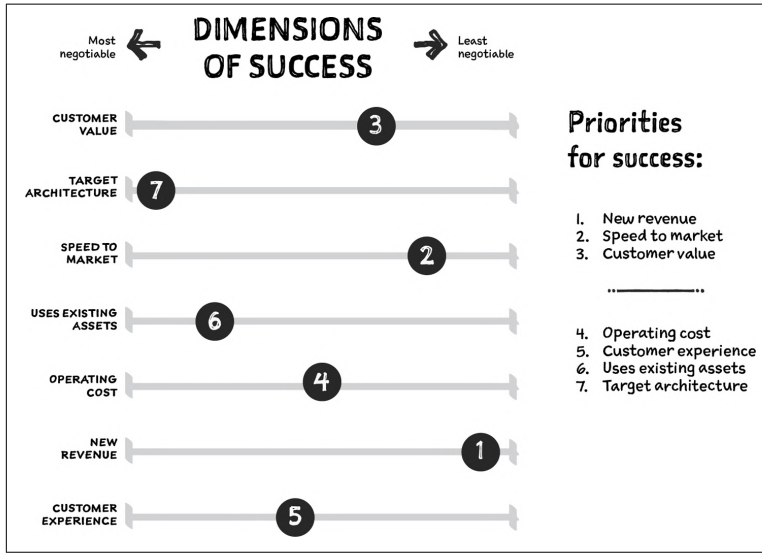


Figure 4-2. Dimensions of success for a digital team

Making Success Measurable

We measure things so that we can make better decisions, and measurement is the only way we can answer the question: *is it working?*

> If a measurement matters at all, it is because it must have some conceivable effect on decisions and behavior. If we can't identify a decision that could be affected by a proposed measurement and how it could change those decisions, then the measurement simply has no value.[9]
>
> —Douglas W. Hubbard

We need to consider how valuable a piece of information will be in the future. Will it help us to make decision? If we hit the number or reach a specific target, might we then stop further work because we've done enough? What if it moves the wrong way; what will that mean, and what might we do about it?

9 Hubbard, Douglas W., *How to Measure Anything: Finding the Value of Intangibles in Business* (Wiley, 2014).

It's not that we need to predict all potential possibilities or put in place a precise system of measures that automates our decisions. Rather, we need to think about the indicators that will be useful for making some sense or meaning. What can we observe, that will help in making decisions?

Suppose that you're operating an online electronics business, selling everything, from $5 Arduino breadboard kits to high school students, to $50,000 control system components to avionics engineers. You want to make it the best experience possible for customers, and you believe from talking with them that easily finding what they're looking for is the most important thing to them. You have a range of initiatives in play aimed at improving things for customers. What measurements will you take, and how will you use those to determine what's occurring, and what to do about it? Table 4-2 illustrates how to break down goals into useful measures of success.

Table 4-2. Goal-level metrics (source: ThoughtWorks EDGE Playbook)

Goal: customers can easily find what they are looking for	
Measurement	**Comment**
Customers will be delighted	Too vague
Net Promoter Score (NPS) will increase	Subject to external factors, difficult to attribute to a specific initiative
10 customer features or initiatives completed	Measures output, not outcome
Conversion from results page to product detail page increases by three percent month-on-month	Good!
Basket-size for returning customers increases on average from quarter-to-quarter	Good!
Average time-on-site per transaction reduced by six percent quarter-to-quarter	Good!

With a clear understanding of what success looks like and how it will be measured, now it's time to choose initiatives that we believe are most likely to achieve the outcomes.

Value-Based Prioritization

The following prioritization techniques offer fast ways to compare initiatives relative to one another.

Relative prioritization using dimensions of success

Figure 4-3 shows relative prioritization using two dimensions of success for comparing initiatives.

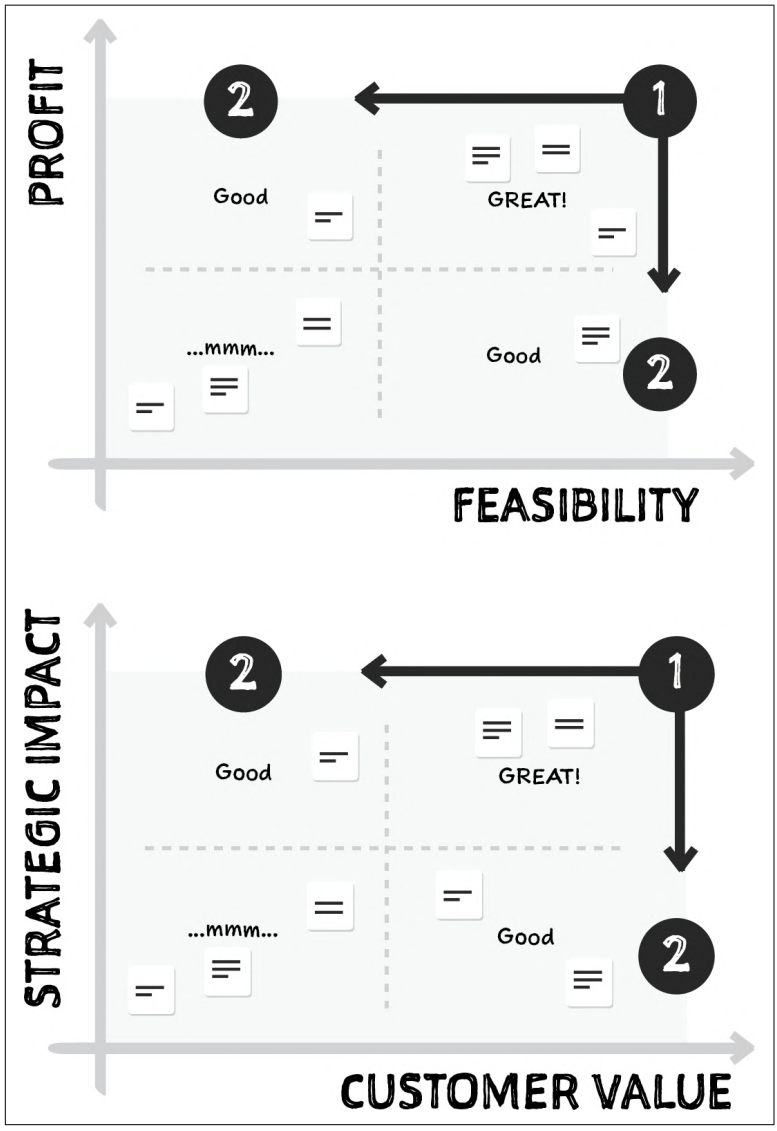


Figure 4-3. Relative prioritization using dimensions of success (source: ThoughtWorks EDGE Playbook)

You're not restricted to using just two dimensions; it is just a common way to visualize competing concerns. In the case of the top diagram, initiatives are ranked relatively, according to their perceived ability to generate profit and the feasibility of implementation. In this case, the initiatives in the upper-right corner would be prioritized, because these have the highest profit potential and are the most feasible.

This kind of prioritization helps to reduce a large number of initiatives into a smaller set on which to focus.

Relative prioritization using weighted average

This prioritization framework uses a weighted average of the dimensions of success to identify the highest value initiatives.

By ranking and selecting the most important dimensions of success and then applying a weighting, we can calculate a relative value of all initiatives in a given collection.

Using our ranked dimensions (see Figure 4-2), let's focus on the first three: *new revenue*, *speed to market*, and *customer value*.

Let's assume that *new revenue* is the most important, with *speed to market* and *customer value* weighted the same, equating to 60, 20, and 20 percent weighting, respectively.

Now we can use the arithmetic in Table 4-3 to calculate the weighted average score for each candidate in a list of initiatives.

Table 4-3. Calculating weighted average to identify the highest value candidate (source: ThoughtWorks EDGE Playbook)

$(Rank_1*Weight_1 + Rank_2*Weight_2 + ... + Rank_n*Weight_n) / \#\ dimensions$				
	Ranked dimensions of success			
	New revenue (60% weight)	Speed to market (20% weight)	Customer value (20% weight)	Weighted average (smaller is better)
Initiative A	1	1	3	.47
Initiative B	2	3	1	.67
Initiative C	3	2	2	.87

A simple weighted average is derived by multiplying the rank by its weight for each initiative, and then dividing by the total number of dimensions being measured. A lower number is better overall, so

Initiative A ranks the highest, Initiative B ranks next highest, and Initiative C is last.

Relative prioritization using CD3/WSJF

CD3 stands for *Cost of Delay Divided by Duration*. This method is also known as WSJF, meaning *Weighted Shortest Job First*. *Cost of delay* is about what an organization stands to lose until the work can be completed and value capitalized. For all intents and purposes, *cost of delay* is about *business value* and *time* for a given initiative.

This kind of prioritization is a hunt for the smallest, highest value initiatives. *Small* is critical. We know from Lean manufacturing and theory of queueing that *small batch sizes* are fundamental for optimizing flow.[10] In product development, we make initiatives as small as we can while still having value. This reduces cycle time, allowing value to flow more continuously. Smaller chunks of work also equate to more flexibility because small commitments mean small losses if an initiative is failing or needs to be abandoned for some reason.[11] This all makes diversification of investments or *optionality*[12] easier to achieve.

So, we need to understand something about the relative size or duration of an initiative and its potential value. Then, we're able to calculate a CD3 score and compare it with others to identify the smallest, highest value candidates.

Back to our example, Table 4-4 shows how CD3 is calculated.

10 Donald G. Reinertsen, *Principles of Product Development Flow* (Celeritas Publishing, 2009).

11 This is especially helpful in combatting *sunk cost fallacy*, a common bias in which the cost of stopping is perceived to be too great, because so much has already been invested.

12 Jason Blum is one of the most profitable Hollywood producers today, and he doesn't do it by making megabudget blockbusters. His studio, Blumhouse productions, exploits optionality by making many small bets on microbudget films and managing costs like a hawk. Some of those films succeed, and they're some of the most profitable in the world. Tune into NPR's Planet Money episode no. 650 to hear the full story on how Blumhouse has the smallest budgets but the highest profits in Hollywood.

Table 4-4. Calculating CD3 to identify the smallest, highest value initiative (source: Andy Birds)

(Potential revenue + Time to market + Customer value) ÷ Duration						
	Ranking					
	Potential revenue	time-to-market	customer value	Cost of Delay (last 3 cols)	Job size estimate (effort)	CD3 score (higher is better)
Initiative A	1	1	3	**5**	1	5
Initiative B	2	3	1	**6**	0.5	12
Initiative C	3	2	2	**7**	3	2.33

In this case, Initiative B is ranked the highest. It has a moderate cost of delay, and a short estimated duration, meaning we can achieve value quickest by doing this work first. All of these prioritization techniques help us to select high-value work and get it done quickly. It's not just about time, and it's not just about value. How much we do (*batch size*), when we do it (*sequence*), and how much work is happening at the same time (*concurrent work in progress*) matters, too. Figure 4-4 shows how limiting work in progress, sequencing work, and reducing batch size can result in earlier realization of value, given the same constraints of time and effort.

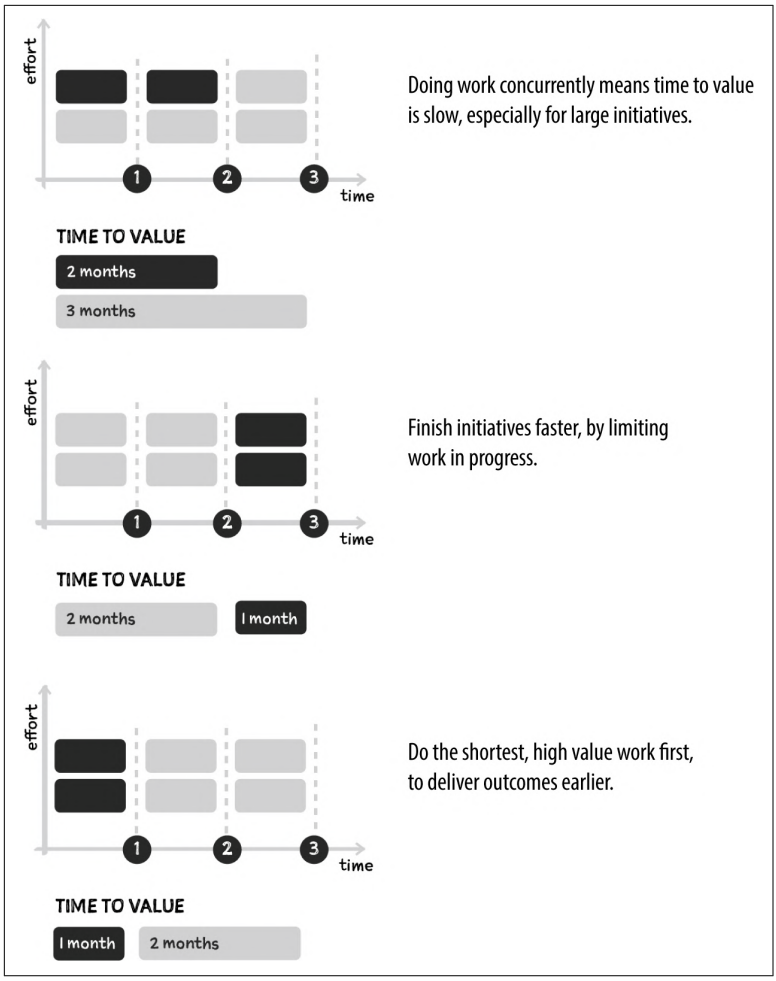


Figure 4-4. Two initiatives of different sizes can be sequenced to reduce time to value (source: Lean Enterprise[13])

Conclusion

When people are empowered with true autonomy, and aligned to purposeful missions, they're not only more motivated, but also able to overcome challenges and achieve outcomes in ever-changing situations. Mission command is a protocol for leading teams in this

13 Jez Humble, Joanne Molesky, and Barry O'Reilly, *Lean Enterprise* (O'Reilly, 2014).

way, and it melds beautifully with scientific thinking and deliberate practice of the Lean mindset.

To do this well, clarity of purpose and an understanding of success and how it's measured is paramount. Visual management techniques offer a variety of ways to align to purpose, whether it's an entire organization, a portfolio of initiatives, or one product team on a mission.

Even though purpose gives direction, teams need to prioritize value and measure success to find their way. Measurements are the signposts of progress that we use to decide what to do next. This, along with value-based prioritization helps teams to maximize outcomes within the constraints of a given system.

Predominantly, it's the Lean mindset influencing our decisions as to what to do, when, and how to adapt our strategy. The ever-adapting nature of Agile delivery plays a supporting role by being an enabler, not a constraint to change. Where Lean has scientific and critical thinking covered, Design Thinking provides the creativity needed when exploring new challenges, as the situation changes.

CHAPTER 5
Delivery Is Still an Experiment

This chapter is about all the ways that Agile delivery enables business agility. We explore how DevOps and Continuous Delivery (CD) enables endless change, making it possible to run continuous experimentation without dramatic consequences. Then, we look at how delivery teams embrace the unknown at scale by using practices like evolutionary architectures and microservices. We'll see similar patterns of thinking as explored so far, but this time applied to *how we build things* instead of *what things to build*.

Until now, we've focused on customer value, identified opportunities, tried many options, embraced a learning mindset, and refined our vision and strategy to get there. You'd be forgiven for assuming that this means building working solutions at scale is merely an engineering challenge. We have *validated* what to build, now we just need to build it.

Wrong.

Software is still an experiment—just a really expensive one. Having delivery teams writing code and deploying working software in no way suggests that things won't change. But it does mean we're probably spending a lot more money to learn what is and isn't working.

So, we need ways of building things that are dynamic and can adapt to change. This isn't just about *pivoting*. It's also about scaling and evolving solutions over time. If we accept that today's solution might be different than tomorrow's, we want to create it in a way that:

- is fast and economical to meet immediate needs; and,
- doesn't constrain our ability to meet future needs.

> A good predictive project will go according to plan, a good Agile project will build something different and better than the original plan foresaw.
>
> —Martin Fowler

Another complication is that people generally aren't very good at communicating what it is that they require or desire until it's real. "I'll know it when I see it," says the stakeholder. For software, *when I see it* often means *when I experience it*. Prototypes can help by giving a tangible demonstration of concepts and ideas, but often it's not until a working solution has been completed and deployed that people can articulate their thoughts about how it needs to be different.

Because it's intangible, people assume that software is changeable. "We're not building houses," they say. That's true. The problem comes when people conflate *possible to change* with *easy and cheap to change*. Although that can be true, it doesn't happen by accident. Being able to respond rapidly, gracefully, and economically usually requires a change in *how* work happens, along with investment in the infrastructure needed to work that way.

Your team is about to deliver a solution that takes the organization into a new geographic region. Management consultants have postulated, estimated, and modeled how you're going to win. The launch date is locked in stone, and can't be moved without blowing millions already spent on media for the campaign.

In the kick-off, estimates from engineers come in three times higher than expected, due to big assumptions that turned out to be wrong. Looking for opportunities to reduce scope, it's clear that marketing, product development, and commercial stakeholders all have a different interpretation of the solution they want.

Designers and analysts work together to rough-out the product architecture, and create a basic prototype. The executive team scoffs, "That's not what we asked for," sending the teams back to the drawing boards.

Time goes by.

Under pressure to make deadline, technical work begins in earnest, and the team complies with a request to make a *clone* of the existing solution and modify it for the new requirements. No agreement can be reached on product design.

The release date comes and goes. Six months late, and three times over budget, the bare bones of the solution are launched into market.

Customer response is lukewarm. Acquisition goes through the floor. Nowhere near what was initially forecast.

Three months later, new federal legislation is passed, and the market regulator mandates a slew of changes. The decision to clone the existing solution means that two instances now need to be changed in parallel—there isn't enough time or money for a ground-up rebuild, so the team is stuck with twice as much work to make the change.

The goal of great software teams is to handle this kind of situation better. Let's look at ways that we can do this.

DevOps and CD

DevOps is a *portmanteau* of the words development and operations. And, like its name suggest, it blends the activity and responsibility of development (building software systems) and operations (running software systems). Why did this become a "thing"? Because doing them separately has a negative impact on an organization's ability to react. When a separate team builds software, and a different team operates it, bad things happen.

Things take longer because there needs to be a hand-over of responsibility, which, in a siloed, risk-averse culture, is likely to include forensic scrutiny and quality checks for the purposes of ass-covering, before an operations team will accept the work. That's understandable, because it's the operations team that is bound to Service-Level Agreements that might require a server to be restarted at 3 a.m. on a Saturday morning.

Things cost more to change, because development teams, often under pressure to meet a launch deadline, might be focused more on completing a working product, than building something that's scalable and easy to work on after launch. It's easy to scald teams for such

behavior, but that criticism is usually misguided. This happens not because of bad people, rather it's a symptom of how organizations are structured,[1] combined with unrealistic demands, like top-down deadlines that have no relationship with reality.

When things fail, it's more difficult to recover. That's because the ops team that is running the system didn't build the system. The intricacies of its inner workings, the idiosyncrasies of its configuration, and the built-in fragility caused by the initial race to hit a deadline makes for a tough puzzle. This has the potential to cost organizations dearly. Like the four-and-a-half hour outage that grounded more than 400 flights for American Airlines while it restored its reservation system after an IT glitch. IT systems are often more expensive to run than they are to build, with *Total Cost of Ownership* dwarfing the initial project cost. Yet IT organizations are often optimized for delivering software systems, not running them.

Enter DevOps.

The DevOps movement says "you build it, you run it," and "you break it, you fix it." It's all about merging development and operations together. Collaboration, automation, and continuity are key themes for DevOps. Teams automate things, specifically the *build pipeline*. Automated build pipelines solve the dev-to-ops handover problem. There is no handover. Instead, all changes are integrated continuously, in keeping with an agreed protocol of automated steps, tests, and decision points. Working in this way means that deployments—to any environment, including live production—can be made at any time. Continuous Integration (CI) and CD are Agile development practices that give businesses the ability to release software continuously. It makes a lot of sense when you think about it:

- Small chunks of value can be released often
- Releasing software to customers is now primarily a business decision, not a technology decision

1 Conway's Law.

Making It Small and Often

Being able to release *small batches* of work, frequently or continuously, is game changing. Think of the poor team in the example at the beginning of the chapter. It took many months and much more time and money to get its solution to market. It was a *big bang* release. And, the team learned too late that customers didn't want it. That's an expensive lesson learned.

Small chunks mean a shorter cycle-time from concept to cash. It means organizations have more flexibility to decide what to invest in. It means they learn faster and reduce risk in their investments. In short, continuous delivery of software is an enabler of the Lean management style described in the previous chapters. And, from a product development point of view, it means that design can be customer-led, starting with an early release of a usable product, and iterating toward the best solution based on real feedback, as demonstrated in Figure 5-1.

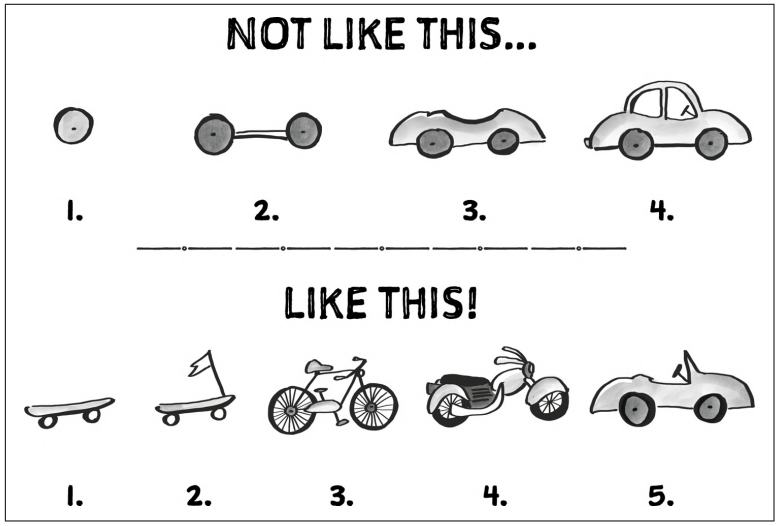


Figure 5-1. Release value early and iterate toward the best solution

Freedom to Decide When and How Much to Release

When changes are continuously integrated and quality assurance tests are happening all the time, there's nothing left to do but press the big red button, as depicted in Figure 5-2, when the organization decides that it's time to go live.

Figure 5-2. Deploy!

This is awesome in a few ways. It's far less risky. What CD does is "bring the pain forward," to make difficult things easy, by doing them more frequently. When teams do this, deployment complexity and risk is reduced because the wrinkles have already been ironed out *before* deployment time, making things go more smoothly than a big-bang release, as illustrated in Figure 5-3.

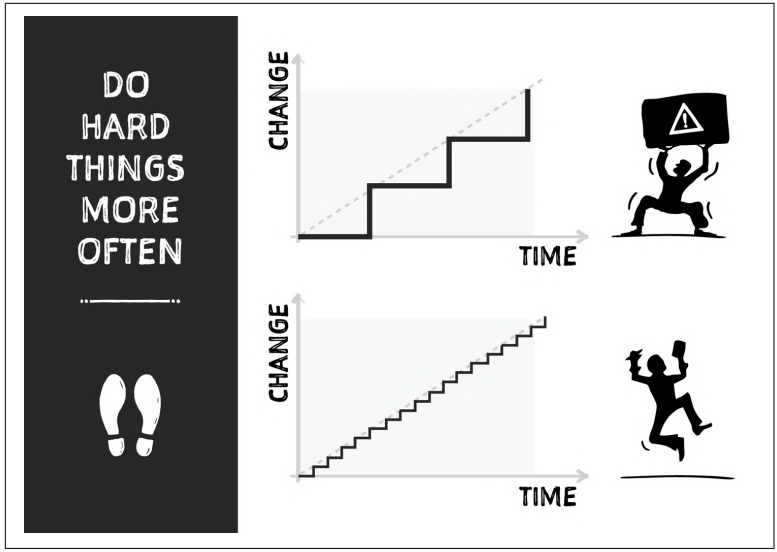


Figure 5-3. Do difficult things more often to make them easier

Controlling when and how much to release is a powerful choice. One indicator of high-performing organizations is the frequency at which deployments are made:

> High-performing organizations decisively outperform their lower-performing peers. They deploy 200 times more frequently, with 2,555 times faster lead times, recover 24 times faster, and have three times lower change failure rates.
>
> —Puppet, State of DevOps Report 2016

But what's most important is *choice*. Every change need not be released instantly. Sometimes, product teams want to release certain elements together. Like a group of new product features that correlate with a point-in-time marketing campaign. This scenario requires a degree of control and coordination.

On the other hand, many changes—even product features—are best deployed incrementally, frequently, and in small pieces. It's just less risky and easier to adjust when things go awry. It's better for everyone. Humans tend to resist change, even when the change equals improvement. So, breaking change into smaller bite-sized chunks makes it easier to swallow:

> Most redesign launches result in an initial dip in conversion [...] but Marks & Spencer's online sales plunged by 8.1% in the first quarter following the launch of its new website.
>
> —Forester Research, Make This Website Redesign Your Last

DevOps and CD allows choice. No longer must organizations batch all changes in monstrous three-monthly releases, and then hold on tight, hoping for the best when it's time to go-live.

Evolutionary Architecture and Emergent Design

These approaches help us to efficiently build today's technology solution without constraining or complicating future work to extend or change it.

Evolutionary architecture is about understanding the principal characteristics of a system—sometimes called *fitness functions*—and defining the target architecture in a way that accommodates those needs, yet without locking everything down. The goal is to *not* make decisions that might restrict options later, unless it's absolutely necessary.

The initial target architecture gives just enough guidance for teams designing technology solutions. Emergent design describes how a solution comes to bear, based on team-level choices like *Are we using AngularJS or ReactJS*? By working within the boundaries of the current architecture, choosing preferred options, and just getting started, the design emerges as we explore what works and what doesn't.

Both evolutionary architecture and emergent design (see Figure 5-4) shift the responsibility of decision making to the teams closest to the work. And, decisions are delayed until the *last responsible moment*, allowing flexibility to respond to new information as we learn it.

Sound familiar? This is the same mental model implied by mission command and autonomous teams that were introduced in Chapter 4, only this time we're talking about *how* it's built, not *what* is built.

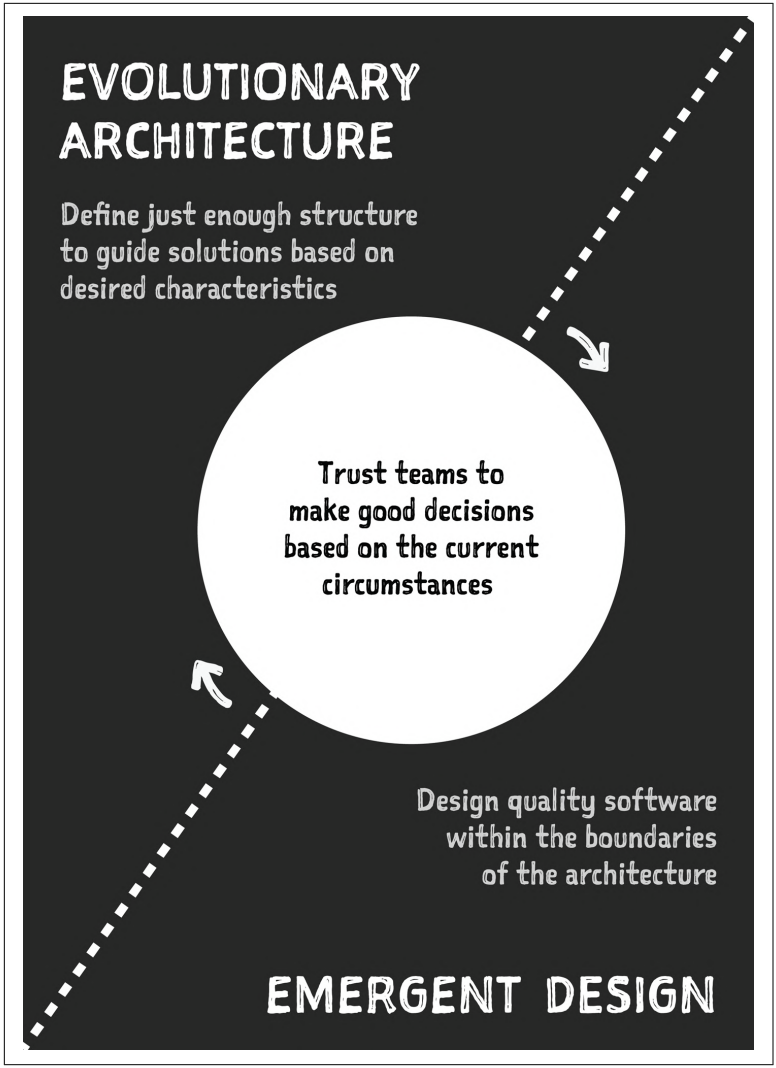


Figure 5-4. Evolutionary architecture and emergent design

Small, Modular, and Service-Oriented

Microservices architectures is a broad topic. Instead of diving into the details (for that, I recommend that you read Martin Fowler and James Lewis' paper, or Sam Newman's book *Building Microservices*[2]),

2 Sam Newman, *Building Microservices* (O'Reilly, 2015).

let's instead highlight the benefits of microservices, and how it fits with everything else we've talked about so far.

Using microservices is a modular approach to building evolutionary architecture. It's how we break down enormous, all-encompassing blobs of server-side computational logic, into smaller, more independent, and easier-to-handle components. Doing this is beneficial in lots of ways. From an organizational agility point of view, here are just some of the benefits:

Each service is highly decoupled from other services
: They're self-contained, thus technical changes in one service (for example, an internal data structure change) cannot affect another service, making it easier to make changes in isolation without risking catastrophe of unintended consequences, or difficult coordination with other teams in order to make a change.

Infrastructure is not shared
: Developers have freedom to make the most appropriate technology choices for the task at hand. This means that teams can choose simple tools for simple problems, and complex tools for complex problems. This is more economical and flexible.

Services are oriented around business functions, not job functions
: This means that it's easier for one cross-functional team to manage a given service, instead of distributing and coordinating effort across many teams to get work done.

So, microservices is all about choice, flexibility, and responding well to the needs of the organization, by decoupling services, democratizing infrastructure choices, and orienting teams to business functions.

Conclusion

These ideas and approaches coalesce into a competitive advantage for organizations that embrace them. Microservices and evolutionary design enable organizations to delay critical technology decisions until the *last responsible moment*. Making decisions later means that teams can explore what makes sense and what doesn't while pursuing the desired outcome. Over time, our threshold of knowledge becomes higher because we've taken some action, not just theorized, and we obtain more information to make a better decision.

Because services are isolated from infrastructure (modular) and small in size (micro), they're more nimble and flexible. This enables real-time experimentation. It's plausible—even trivial, given the right conditions—for a team to build two complete solutions in parallel, and test them over time, with live data to determine which product approach wins out.

DevOps and CD is how we build more robust software in a way that's less risky to deploy, faster to recover from failure, and cheaper to change. This helps organizations become more resilient to changing conditions. It also gives greater choice about when and how much to deploy. It's the mechanics of software delivery that makes experimental product development a reality.

It's how technology enables the art of the possible.

Closing Thoughts on Design Thinking, Lean, and Agile

The mindsets of Design Thinking, Lean, and Agile have different origins, helping us to think in different ways and to solve different problems. We've explored how these come together, complement, and overlap one another. Design Thinking brings discipline to how we frame problems, consider the customer, and explore creative solutions. Lean thinking brings discipline to how we learn, make decisions, and coordinate efforts on our quest to achieve our goals. Agile is how we build technology solutions that evolve and adapt as we learn and respond to changing needs that emerge from taking action.

There is no one correct way, nor is one single mindset enough. But all together, elements of each mindset help us to find our way forward.

The four steps of actionable strategy describe how these mindsets come together in practice. It's the scaffolding that frames the work we do. The glue that brings many ways of thinking together as one.

Although we've included practical methods and tools, along with some models to guide how we think and act, making it work is about people. Instead of focusing on process, we ought to challenge how we think, try new ways of doing, adopt the things that work, and learn from the things that don't. This will be different for

everyone. Success is about how we develop new *ability*, *learn* by doing, and *adapt* to what is learned.

If there's one thing to do after you finish reading this, let it be something new. The best way to build new ability is to try something, see what happens, and adapt as you learn.

Acknowledgments

So many terrific people contributed to this, my first published work. I'm forever grateful to the following people, who gave interviews, provided feedback, contributed content, and helped me refine my thinking: Alan Grimes, Amy McCleod, Andy Birds, Dan McClure, Danelle Jones, Darren Smith, Gary O'Brien, Ian Kelsall, Ian Cartwright, James Lewis, Kathy Buttler, Kirk Kinne, Kraig Parkinson, Lourenço Soares, Meg Blake, Mike Tiffany, Natalie Hollier, Neal Ford, Nick Byrne, Nick Thorpe, Pat Kua, Peter Gillard-Moss, Rebecca Parsons, Richard Glew, Rodd Messent, Rujia Wang, Steve Duesbury, Sue Visic, and Tiago Griffo. Thanks also to the editors at O'Reilly Media, Mary Treseler, Angela Rufino, and the production team.

I'm enormously thankful to Barry O'Reilly, who encouraged me to do this in the first place, helped me get unstuck many times during six months of writing, and from whom I've learned a great deal as a colleague—and as a friend.

To Sara Michelazzo, who created all of the fantastic illustrations, grazie mille.

Thanks to ThoughtWorks for being brave enough to take an experiment when you hired me in 2012. The opportunity to work with inspiring colleagues and incredible clients has shaped much of the thinking in this publication.

So many thanks to Ali Radloff, beautiful wife, and my partner in life, for your endless support during my stubborn, myopic pursuit to get this done. This would not have been possible without you.

And, finally, to my parents, Richard and Cheryl, and siblings Paul, Natalie, and Tim. You taught me that something worth doing is worth doing well, and that nothing worthwhile comes without effort.

About the Author

Jonny Schneider creates digital products and services that balance business outcomes with customer needs. He's a leader and practitioner in product strategy, design, customer research and software delivery. Jonny helps companies by bringing a pragmatic, collaborative, and interdisciplinary approach to customer development and problem solving. He consults globally with all kinds of companies in financial services, travel, retail, telecommunication, education, government, science and research, and human welfare.

Jonny lives in Melbourne, works with ThoughtWorks, and loves a nice brogue. He's on Twitter @jonnyschneider and blogs at *jonnyschneider.com*.