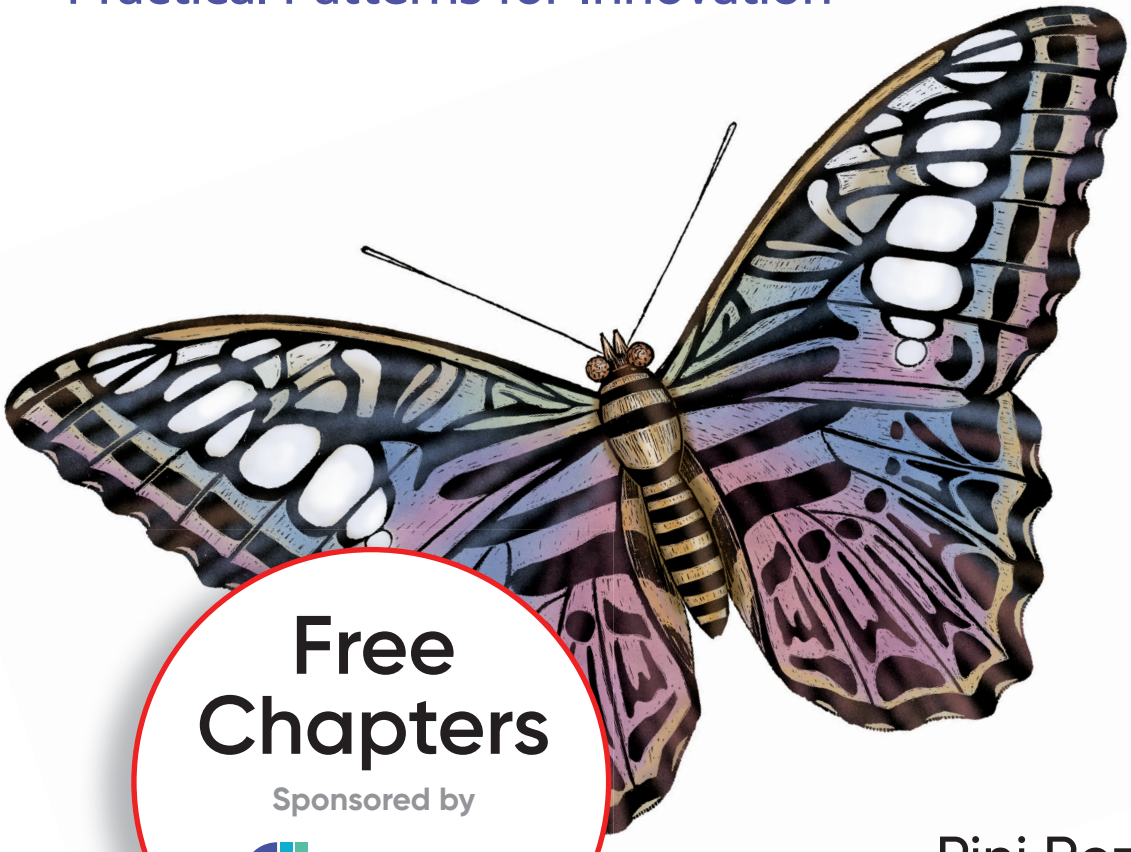
O'REILLY®

Cloud Native Transformation

Practical Patterns for Innovation



Free Chapters

Sponsored by

Container Solutions

Pini Reznik,
Jamie Dobson &
Michelle Gienow

Container Solutions

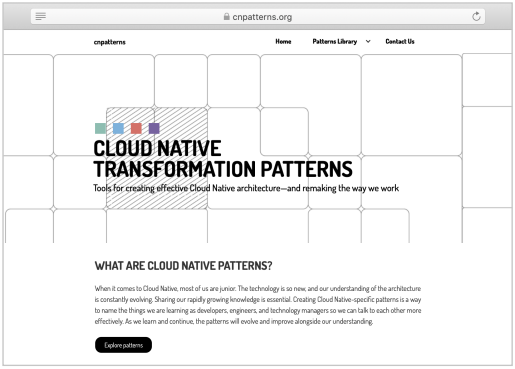
Cloud Native transformation can be risky.

We can guide you—and help you deliver value, fast.

Your company has always handled tough challenges. But today is different. There's a new generation of competitors on the rise. They can serve needs customers don't even know they have—and serve them blazingly fast.

That's the power of Cloud Native. But the technology is new, and ever more complex as you move ahead. Very few engineers know how to manage it, and the options are endless. It's easy to make wrong choices that waste time and resources.

At Container Solutions, we have been doing Cloud Native since before it had a name. Our new O'Reilly book, "Cloud Native Transformation: Practical Patterns for Innovation," includes some of the best lessons we've learned along the way. We are also sponsoring a new online community, **CNPatterns.org**, for creating and collecting new practices and strategies.



CNPatterns.org is a new online community for finding and sharing Cloud Native patterns.

Our experts guide companies through Cloud Native transformations from initial strategy to running their new systems self-sufficiently. Our expertise has served Cloud Native projects for companies like Google, Shell, and Adidas.

We can do the same for your business.

It starts with our Cloud Native Transformation Workshop, a fun, hands-on way to learn CN fundamentals using patterns playing cards. Led by one of our CN experts, participants collaborate to architect basic transformation strategies while exploring how to evolve culture and processes along with your new technology. You don't need to be an engineer to participate!

If you're ready to move forward, our on-site, three-day Cloud Native Assessment helps show you where you stand now, and where you need to go next. Afterward, Container Solutions can also help you think through your strategy, collaborate on small-scale experiments to eliminate risk as you design a Cloud Native platform, then build and run it self-sufficiently. In most cases, we can help you deliver a new Cloud Native application to production within four months—and deliver value to customers faster than you ever imagined possible.

Container Solutions' patterns cards feature more than 70 Cloud Native patterns to help you map your own custom transformation.

To get the full picture:

- Buy the book.
- Check out CNPatterns.org for the latest information.
- Get a deck of our patterns cards to jump start your understanding of CN concepts: info.container-solutions.com/cnt-cards
- Invite us to deliver a half-day CN Transformation Workshop. Each participant gets a copy of our book and a deck of patterns cards: info.container-solutions.com/training/cn-transformation-patterns-workshop

Cloud Native Transformation

Practical Patterns for Innovation

This Excerpt contains Chapters 6 and 11 of the book *Cloud Native Transformation*. The complete book is available through O'Reilly Online Learning and other retailers.

Pini Reznik, Jamie Dobson, and Michelle Gienow

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Cloud Native Transformation
by Pini Reznik, Jamie Dobson, and Michelle Gienow

Printed in the United States of America.

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

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

Editors: Michele Cronin and Chris Guzikowski
Production Editor: Nan Barber
Copyeditor: Jasmine Kwytin
Proofreader: Kim Wimpsett
Indexer: Judy McConville
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

December 2019: First Edition

Revision History for the First Edition
2019-12-03: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781492048909* for release details.

This work is part of a collaboration between O'Reilly and Container Solutions. See our statement of editorial independence.

978-1-492-04890-9

[LSI]

Table of Contents

Foreword

A few years ago, I was in Amsterdam for Dockercon Europe and met a very enthusiastic British guy with a northern accent, Jamie Dobson. He seemed to be running a company in Holland, but it wasn't at all clear what they did. A bit later that company became Container Solutions and expanded to London, and I kept running into Jamie, Pini Reznik, and Adrian Mouat (and eventually Anne Currie) at other conferences. Then the Container Solutions crew came up with this crazy idea to run a conference in a dockside warehouse in Amsterdam, one that mixed software with a kind of Burning Man vibe. It was called Software Circus.

Outside there was a 30-foot-tall fire-breathing robot called Kevin[1] and a whole pig roasted on a spit for lunch. I was the opening keynote speaker, although I had no idea who would come and what they would want to hear about. Over 400 people turned up, and I improvised the talk with no slides, told stories, got suggestions from the audience, and the projected Twitter feed became an effective crowdsourced substitute for a presentation. Afterward, Kelsey Hightower, who was giving the keynote on the second day, told me he hadn't ever previously thought of doing a talk with no slides, but he was now inspired to have a go. Kelsey then famously gave an entire talk on container scheduling while displaying himself playing Tetris. It was one of the more memorable conferences I've attended, but there were no videos of the talks; you had to have been there!

The people at Container Solutions think differently, don't play it safe, and seem to have a lot of fun along the way. They've worked on some very interesting projects over the years, and this is an unusual book; it's a collection of stories and a pattern library. The stories are entertaining and illustrate the same common patterns I've seen around the world, as companies struggle to adopt DevOps, product-based teams, microservices, continuous delivery, containers, and serverless. The pattern library is best approached as a cross-indexed random access collection. You could do

1 Because Kevin is an entirely inappropriate name for a fire-breathing robot that folds up into a 10-foot cube.

a linear full table scan by reading it through, but it's better to pick an interesting starting point and walk the graph by following the cross-references to related patterns. The individual patterns contain a lot of great detailed advice and aren't afraid to be opinionated. I agree with most of the opinions and find that they reflect the real-world experience that is often written out of books that play it safe with generic answers.

From my experience I'd like to provide three core ideas that this book illustrates in detail. First, most of the customers I meet think they have a technology adoption problem, when in reality they have an org-chart problem. The most successful orgs are made up of many small independent business teams communicating via clear APIs. If that's your org chart, you will inevitably build a microservices architecture, a trick known as the Reverse Conway Maneuver.

Second, the single most important metric for executives to focus on is time-to-value, which can be instrumented most easily as committing to deploy latency for each team. Many companies can't innovate because they can't get out of their own way, and short time-to-value (target less than a day) releases pent-up innovation from your teams. Third, there is far too much time spent in analysis paralysis, discussing technology variations and arguing about how best to architect solutions.

The patterns in this book will be a helpful influence to center these arguments, speed up architectural decisions while reducing the risk of making a wrong choice, and ultimately create more commonality as the patterns that work replace low-value architectural divergence. The stories remind me of Gene Kim's books *The Phoenix Project* and *The Unicorn Project*, but I haven't seen storytelling paired with such an extensive and detailed pattern library before. It's a really compelling combination, doing a terrific job of illustrating the patterns themselves and showing how they fit together. It's also a lot of fun to read.

— Adrian Cockcroft
Cloud architect and engineer
November 2019

Prologue: Evolve or Else

Cloud native is more than just a technology or tool set. It is a philosophical approach for building applications that take full advantage of cloud computing. This new paradigm requires not only embracing new technology, but also a new way of working—which makes going cloud native an extensive undertaking. The payoff is immense, but in order to reach it an organization must evolve not just its tech stack but also its culture and processes.

The story of WealthGrid, a typical enterprise undergoing a cloud native transformation, illustrates this process from start to finish. First, we will look at the factors inspiring WealthGrid's decision to pursue a cloud migration. Then, as we follow WealthGrid's migration path, we will witness the problems it encounters along the way—ones that are commonly experienced during the transformation in companies both large and small. Ultimately, we will see how to use patterns to overcome—or even better, prevent—those problems.

Welcome to WealthGrid!

Meet the organizational protagonist of our cloud native transformation tale: WealthGrid, a mid-size financial services company. WealthGrid is a fictional composite of real-world companies we have worked with over the past few years, when we were called in as consultants to facilitate new cloud migrations—or to rescue failing ones. We created this composite in part because we must maintain our clients' confidentiality, but also so we can illustrate the most common problems companies encounter. Thankfully, no single transformation initiative is likely to endure *all* of these troubles, but nearly all initiatives will encounter at least some of them.

WealthGrid's organizational chart looks a lot like most mid-size companies: functional and top-down. In other words, it's your traditional business setup, with the C-suite and board at the top followed by other senior management, middle managers (including project and/or program managers), department heads, and so on, down the line. Structurally the company is divided into traditional departments like IT,

marketing, finance, human resources, and operations based on functional role in the organization. There are specific standards, policies, and practices to govern how things work as well as every decision the business makes. Among themselves, teams may have taken on some Agile methods (like Scrum), but WealthGrid itself is fundamentally a classic hierarchical organization following a Waterfall software development approach (from here on out we will shorten this to "Waterfall hierarchy").

Overall, WealthGrid is quite rigid in what each specific department is designed, and permitted, to do for the company. Formal and well-documented channels exist to facilitate communication between them. Since each department specializes in its specific area of responsibility, there are numerous handoffs between the many engineering teams working on any given project; project managers are in charge of coordinating between them. Though the company's hierarchical structure and highly structured communication protocols may sound inflexible and maybe even intimidating, they do confer advantages. At WealthGrid, the chain of command—whether short or long—is always crystal clear. Every team's function is explicit, and their responsibilities are clearly spelled out, which means they are optimized for proficient productivity.

Sure, this model might hinder creativity and experimentation, but WealthGrid has been around for a long time and this is the way it's always worked. It takes a lot of managers to keep a well-oiled machine like this one, with so many people and projects to keep track of, running smoothly. Or at least as smoothly as possible.

Sound familiar? Though many companies have taken some steps toward more agile management practices, most mid-size and larger enterprises still function as hierarchies using a Waterfall software development approach. WealthGrid could be almost any company, anywhere. It is presented as a financial company because that industry is particularly representative of the pressures facing many sectors. Competition is high in the financial sector, driving the existential need to evolve and keep pace. Meanwhile, it is populated by a wide variety of companies with strikingly different levels of technological savviness. No matter what your business, though, you are likely facing some, if not all, of the same market pressures.

WealthGrid could be just like your organization: a business in the real world, mid-size or even larger. Not a giant technology company, but big enough to have a real IT department with engineers. And, although yours may not be a software company, you still need to deliver software—as must almost every business these days.

WealthGrid is a well-established company with a good reputation. It's been successful for years, decades even. It delivers solid products and services, and its internal culture is positive and healthy. People enjoy working at WealthGrid. They have competition, of course, but it's not much to worry about—the players are all familiar and competing within a stable and steady market. Thus WealthGrid is financially healthy, with

reasonable (though not huge) profit margins. Overall, it's a good company, with a good market. It's good to be WealthGrid.

Until one day…

A Stranger Comes to Town

> *All great literature is one of two stories; a man goes on a journey or a stranger comes to town.*
>
> *—Leo Tolstoy*

This is the story of how a stranger suddenly appears in WealthGrid's nice, comfortable world—and changes everything.

Not a literal stranger, of course, but a brand-new competitor suddenly entering the financial services market. Someone completely unanticipated, completely different from traditional competitors, and very dangerous. There are three types of strangers who might, so to speak, come to WealthGrid's town.

- **The Upstart.** The first possible stranger is a company like Starling Bank. Starling is a brash newcomer, a UK-based contender bank that launched in 2014 and essentially built an entirely functional bank in a single year. Starling, which is mobile-only (no physical locations, nor even a web-based interface) has grown steadily, adding business accounts with full mobile functionality and seamless integration of services. The bank is approaching half a million customers, most of them young. It's very efficient and very effective. Starling is not big yet but it's growing very, very fast. And, at the time of writing, there are 26 other challenger banks just like this in Europe alone, with a high likelihood of more on the way.
- **The Rogue Giant.** The second potential stranger is a bit more familiar: Amazon acquired a banking license in 2018. No one knows what the company is going to do with it—it's not saying—and there is much speculation that Amazon will be entering the retail banking market. Studies show that, if it does jump in, Amazon could quickly become the third largest bank in the United States, eventually drawing an estimated 70 million customers.
- **The Reinvented Competitor.** The third possible stranger is, in a way, even more well known: a traditional competitor, but one that is rapidly reinventing itself. For WealthGrid this means a traditional bank that is investing heavily to modernize both its technologies and the products and services it offers. Someone like

ING from the Netherlands—a bank that these days doesn't call itself a bank, but a tech company with a banking license.[1]

All three of these strangers are cloud native entities. Starling Bank was born in the cloud and optimized from the start to take advantage of the velocity and inherent scalability that cloud native confers. Amazon is literally one of the inventors of significant cloud native technologies, and ING is well on its way to transforming itself from a traditional bank to a fully optimized cloud-based operation.

Why should WealthGrid worry about any one of these, though? After all, the company has done a pretty good job building its functionality, keeping up with customer demand for online and mobile banking features. What do any of these "new" strangers offer WealthGrid's customers that it does not already also provide?

Stranger Danger

The danger at this point is still somewhat subtle. WealthGrid is a profitable company, and so much of its annual earnings go to shareholders. Some might go back into the company to invest in innovation and research, but if so, it's a tiny portion of profits. The problem is that WealthGrid's technology growth has been pretty much a straight line: the value it is building for its customers is growing, yes, but in a linear way. This is exactly what a company like WealthGrid likes. Linear also means predictable and stable. It's easy to do long-term planning with linear growth. It has worked well for decades, why stop now?

If you are a traditional company, however, and a competitor using cloud native technology advantages comes into your sector, that competitor's growth is not going to be a nice, straight line. It's going to be a steep exponential curve. Figure P-1 shows the comparative growth for a traditional company versus a disruptive newcomer who enters the market later, but with a cloud native advantage.

1 ING CEO Ralph Hamers during a video interview with British financial affairs publication *The Banker* in August 2017.

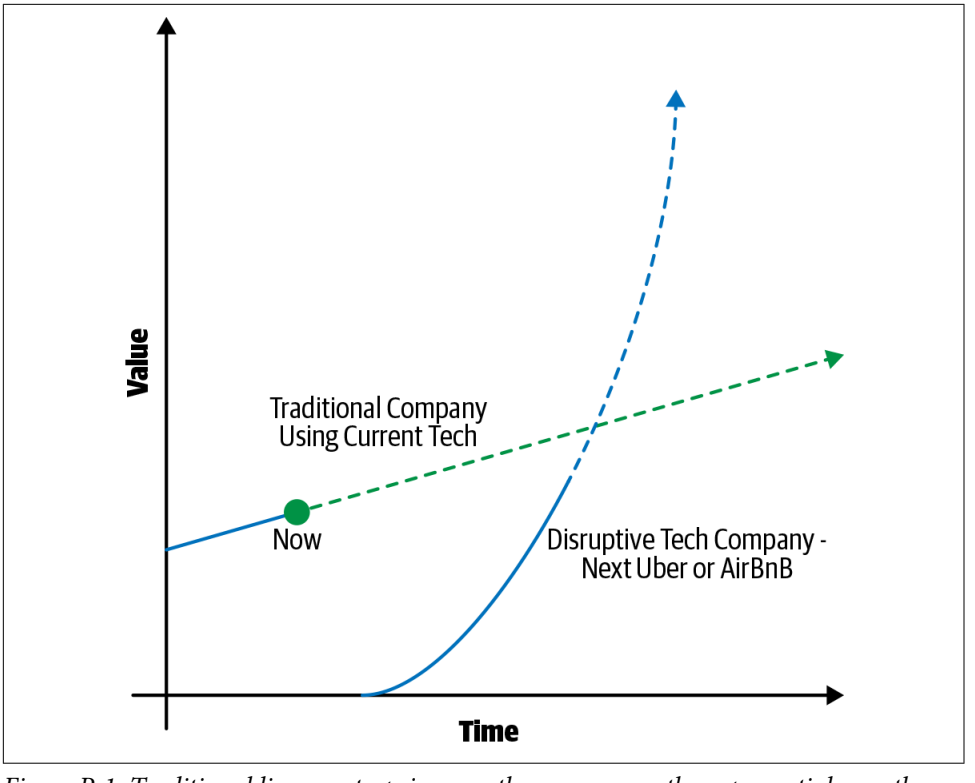


Figure P-1. Traditional linear enterprise growth curve versus the exponential growth curve a disruptive new competitor using cloud native technology will bring

At first the stranger is brand new and very small, taking only a tiny portion of your market. It's easy to feel complacent. Do not be fooled: they are still genuine—and serious—competition!

It is understandable that you and your usual competitors, the traditional vendors in a given market, may not be too worried at first. After all, you still own the majority of the market. But those newcomers will be growing exponentially. They, after all, don't carry baggage from decades of previous development and so are able to operate at a significantly lower cost. Worse, once they do demonstrate (even a little) growth, they will also have access to more or less unlimited investment funding. And they will take this money and plow it right back into ever more technological advancement, driving their growth curve ever steeper and ever higher.

So how are these newcomers so successful so fast? The main reason they are able to so quickly establish themselves in your market is because they deliver faster and more frequently using modern technologies. They have a rapid cycle of building functionality, delivering it to the market, getting immediate customer feedback and, based on

that feedback, delivering even more feature innovations and improvements. They can do this very, very quickly.

Some of these high-performing development capabilities are also used by companies following Agile and Lean practices, and those are definitely a good start. Cloud native, however, was born to take full advantage of cloud computing. For example, Agile teams may do continuous integration, but typically still deliver in one- or two-week sprint cycles. Cloud native teams add continuous delivery to create CI/CD and the ability to do new releases every day, or even multiple times each day.

The technology enabling this serious competitive advantage? Cloud native.

So these newcomers are on public clouds like Amazon Web Services, Google Cloud, or Microsoft Azure, taking advantage of the ever-more powerful services integrated with these platforms. They have optimized their architecture for microservices and containers, and use Kubernetes to orchestrate everything. Those technologies and tools and cloud-centric processes allow companies to move very fast and deliver functionality every day … or even many times a day.

Worse, they aren't entering to compete with you wherever you happen to be right now. Instead they enter the market with a serious head start, thanks to their superior technical position. They use customer feedback to iterate quickly on new features, and then get them in front of users *fast*.

No matter which stranger comes to your town—they may not be there yet, but they are definitely on their way—they will be jumping in way, way ahead. The simple truth is, once they do, you will not be able to catch up. They are going to leave you behind in your own market. You have to start now, before this competitor shows up. Because they are coming—and this way, you can be ready for them.

(If you wait until they are already here, you'll have to contend with the exponential growth curve we showed you in Figure P-1. If this disruptive stranger is already in your town, it very well may be too late for you to catch them!)

At this point you may be thinking, *This won't happen to me. Our sector is completely different, we aren't high tech, disruptors would never bother with us.* And we don't blame you; nobody likes hearing that their world is about to be upended. But think about the taxi business—global in scope, granular in implementation. Who could have imagined the disruption of such a widespread, low-tech industry? Yet Uber and Lyft, using a ridesharing model made possible by cloud technology, have done just that. Airbnb did the same thing in the hotel industry. And in both cases, once these new companies entered, the speed of change in the market was very dramatic.

Cereal Killer

Maybe you're still thinking you'll be OK, that your sector just isn't compatible with that kind of disruption. So another aspect to consider: What would happen to your business if Amazon suddenly bought your biggest competitor? Or even a small one, and then used their serious technological advantages to turbocharge that competitor? No matter what sector they may elect to enter, Amazon brings a massive innovative advantage in IT that traditional companies cannot replicate.

Case in point: in 2017, Amazon made an unprecedented move into the US retail grocery market by purchasing the Whole Foods chain of stores. On the surface, this was an utterly counterintuitive thing for Amazon to do. Supermarkets—you can't find a business more basic than that. They are everywhere, and everyone needs them, but profit margins are historically razor thin. Markets are very much fragmented by region, and studies have consistently shown that food is the one thing the vast majority of shoppers prefer to buy in person rather than order online.

Nonetheless, Amazon is now actively building its presence in the retail grocery world. The company has announced rapid expansion plans, including opening new checkout-free grocery stores in a dozen cities across the United States by the end of 2020.[2] Suddenly there is a serious challenger in a very stable, even stodgy, and low-margin industry that never saw it coming.

Amazon (along with other tech behemoths like Google and Facebook) created the cloud revolution in the first place to serve their online business needs. Now they are completing the cycle by moving into analog infrastructure, which is easy to do, and very available to them: the company has announced that one possible expansion route could be for it to buy up bankrupt supermarket and other retail chains that failed to compete and move new Amazon-run food stores right into that ready-made infrastructure.[3]

Amazon may not be in your market right now. But when it does come, it will be instant. And the competition may well be lethal.

You Say You Want an Evolution

This story is happening in the world right now. Even if WealthGrid is hypothetical, all three of these strangers actually exist.

2 Here is a good overview of how and why Amazon got into the grocery business and their plans for the future.

3 Amazon exploring bankrupt or vacant retail infrastructure to house new brick-and-mortar Amazon Fresh and Amazon Go stores.

Companies are beginning to wake up to the existential threat. They are responding slowly, though, in part because of the lingering belief that cloud native is only accessible to giant tech companies. That belief was true just a few years ago, but now it's seriously misguided. Tiny young companies like Starling are starting small in the cloud and making it work for them. ING, an old-guard financial company, recognizes the existential threat and is seriously working to reinvent its legacy systems. Cloud technologies are maturing and becoming mainstream, and any company can "do" cloud native these days.

Like so many others out there, WealthGrid is a healthy company. It has been successful for years; it delivers a good product, and people enjoy working there. Truly it has done nothing wrong. Inevitably, however, its environment is changing. It is time to evolve, or risk extinction.

In nature, the only reason species ever adapt is to gain a benefit (survival being a pretty major benefit). If there is no benefit to gain, they simply won't change: evolution, after all, is a costly process. This works exactly the same way in the business world. If a company is not exposed to pressure, it also will never adapt.

The catalyst or pressure may not be there yet, but the very high likelihood is that it is coming to your industry. How do you know when just such a disruptive stranger will come to your market?

The answer is, you don't.

Which Evolutionary Stage Are *You* In?

In our experience it takes two to three years to truly undergo a cloud native shift and adjust to this new way of working. After that, it takes five more years to truly internalize the change—you can't succeed in an entirely new paradigm like cloud native without understanding the tools as well as how to transform your organization so it can best apply them.

Smart companies recognize that technology is always advancing, and they strive to innovate right along with it. Those that fail to adapt will fail, period. Some companies, even entire industries, will disappear. This is all part of the evolutionary process, and it will not destroy the world—it will build a better one. Cars did not destroy the world when they replaced horses; they just changed it. People switched from riding horses to driving cars. It's all thanks to technology that we now have things that even kings couldn't afford, or even imagine, 200 years ago.

This may all sound terribly dire, but it's really not. In crisis lies opportunity. If WealthGrid wakes up in time to the stranger's impending arrival and takes steps to adapt, it will emerge a stronger, more resilient and innovative company.

Change is inevitable, and the fittest species respond and adapt accordingly. Let's watch as WealthGrid gives it a shot.

INTERLUDE I

Meet Our WealthGrid Team

Now it's time to meet the protagonists of the WealthGrid story. Jenny, the program manager, is the character we will be following most closely through WealthGrid's journey to cloud native. Jenny is first to realize that her company is facing pressure—that the stranger is coming to WealthGrid's town—and needs to evolve. Even if others have perhaps also recognized this, she is the first to take real action. Either way, Jenny is the catalyst for WealthGrid's cloud native initiative. In later chapters we will also meet Steve, WealthGrid's CEO, who plays an equally important role in the company's cloud native transformation efforts.

So where were we? Oh, yes: a stranger has come to town. WealthGrid is facing pressure, perhaps even genuine existential threat, to adapt to this change in their market/environment.

Jenny has been a program manager at WealthGrid for several years, and she is good at her job. At WealthGrid, as in any mid-size or larger Waterfall or Agile-ish company, program managers are quite literally middle management. They sit squarely between upper-level executive leadership, which is responsible for defining strategy and project initiatives, and the engineers responsible for delivering the actual product. Program managers act as project facilitators and keep communication flowing smoothly between these two groups. Working with the engineers, project managers have detailed boots-on-the-ground insight into a project's progress. In turn they keep upper management informed, only without bogging them down with unnecessary details. Project managers also spend a lot of time interfacing with other middle managers, facilitating project handoffs between WealthGrid's specialized teams, and trying to keep production schedules on track.

Jenny likes her work and finds her position an interesting challenge. Being a good program (or project) manager means having enough technical knowledge to understand the engineers and their work, balanced with enough business knowledge to

explain things to the executives. In short, Jenny acts as a translator between the otherwise fairly separate enterprise and technical cohorts within the same organization. She can explain the business case of a proposed feature in terms that rally the engineers to get behind a new project. She can also respectfully communicate upward, in proper C-suite speak, when a suggested new feature might be impractical, maybe even infeasible, from a technical standpoint.

Thus Jenny's job as program manager requires high exposure to the business side: customer conversations, understanding the market forces that shape the company's path, and involvement with planning and improving internal processes. It also involves daily immersion with the teams themselves, closely tracking the decisions they make and the outcomes they produce. She enjoys this because, like most good engineering managers, Jenny herself is a former engineer. Even though she has moved into a management role, she tries to stay up to date and knowledgeable in the latest trends. No matter which hat she's wearing, though, Jenny's natural ability as an organizer and her solid people skills serve her well.

Jenny's job requires her to manage both downward and upward at the same time—to make things happen, but still make everyone reasonably happy on both sides. But being in the middle also means that she gets pressure from both above and below.

Pressure from Both Sides

One particular pressure that has been subtly building for a while now has to do with WealthGrid's response to the cloud computing revolution. Cloud-based services have been radically altering both how enterprises can deliver products and services and how customers can consume them. Jenny herself is aware of the rapid evolution of cloud native tools and techniques—of course she has heard about containers and microservices. But she doesn't have deep knowledge in this cutting-edge tech. Hardly anyone at the company does since it's so very new.

That doesn't stop her engineering team from badgering her about Kubernetes and how cool it is—most tech folks love exploring the latest and greatest innovations in their field, and her team is no exception. For Jenny, this means every time someone comes back from a conference, they are excited about the possibilities of putting together their next project using microservices architecture, orchestrated by Kubernetes. Heck, she's excited too—she goes to a few conferences herself and has heard first-hand about the power of cloud native to revolutionize a company's delivery model to make it responsive, iterative, and, above all, fast. She has been doing some research, and it seems like cloud native is real and here to stay.

But where could this fit within WealthGrid's well-defined, proficient, and long-standing (i.e., legacy) systems and processes?

Interestingly, this pressure from her engineering team dovetails with pressure that Jenny has recently experienced from above, regarding hiring. Recently, hiring for technical positions at WealthGrid has become surprisingly challenging—surprising because WealthGrid's compensation is at the top of industry pay scales. However, openings simply aren't drawing many good candidates. Concerned, Steve, the company's CEO, ordered some internal surveys to identify how the hiring process might be improved.

The answer was plain: WealthGrid's systems are legacy tech. Nobody just emerging from university with a computer science degree wants to work on boring old stuff. They want new and cutting edge! Worse, while high pay may entice some into applying anyway, these hires don't tend to stay long once they realize that there is little opportunity for professional growth. Pretty much the only way to advance professionally if you are an engineer is to move into management. And forget about developing new skills: WealthGrid is really good at what WealthGrid does, but that is all they do.

Some of the company's executive managers have started mentioning cloud native architecture in other contexts beyond the hiring issue. They have read Gartner and similar publications talking about how the cloud is how modern companies now need to operate their business—so they think, of course, that WealthGrid should do this too. This gets considerable pushback from other managers, though, who say that business is good and there is no need to rock the boat.

So, from her position at the nexus of enterprise and technology, Jenny hears a thousand different voices, each one telling her different things. But it is just this simultaneous, often conflicting view into both worlds that gives Jenny a unique perspective. The business-savvy side of her recognizes the stranger danger that threatens to disrupt WealthGrid's comfortable world. Her technical knowledge and experience give her the confidence to do something about it.

In the end, those thousand voices all seem to add up to the same message: it's time for WealthGrid to make a move to the cloud.

INTERLUDE II

First Attempt at Transformation–Cloud Native "Side Project"

This interlude includes a description of Jenny's first attempt to lead WealthGrid through a cloud native transformation. One of the most frequent mistakes we see is a company trying to treat a cloud native transformation as just another simple technology shift. Instead of allocating the resources necessary to implement a major organization-wide evolution of both tech and culture, the initiative gets treated as just another minor upgrade. It's assigned as a side project, or as just another system upgrade added to the backlog of standard tasks. Since getting started with cloud native can be deceptively easy, this may even seem to work well at first. But the reasons why this is a poor way to approach a transformation begin to reveal themselves as the project progresses. Things rapidly grow more complex, however, and more and more problems arise; meanwhile, the transformation team still has to keep working on existing system, too. Eventually the initiative will grind to a halt, though no one is quite sure why or what happened.

When last we saw Jenny, our technical program manager, she had come to the realization that now is the time for WealthGrid to make the move to the cloud.

She had been experiencing pressure for just such a move from both above and below. The engineering team she oversees really wants to try out new tools and technology. Some of WealthGrid's executive leadership, meanwhile, has been reading about how all modern businesses must go cloud native. Recently, upper management also realized that having cutting-edge cloud tech would help them attract and retain engineering talent.

Jenny agrees with the engineers *and* the C-suite. Furthermore, she had identified a third clear driving force that unites both sides: keeping the company's bottom line strong and healthy. WealthGrid needs to move to the cloud, for sure, and doing this now is key to keeping the company competitive.

She is astute enough to recognize that not only is the imminent arrival of a disruptive new cloud-powered competitor a very real possibility, but that if you wait until they show up in your market and begin showing significant strength, it could very well be too late for you to catch up. Remember our exponential versus traditional growth graph? Take a look at Figure I-1 as a reminder.

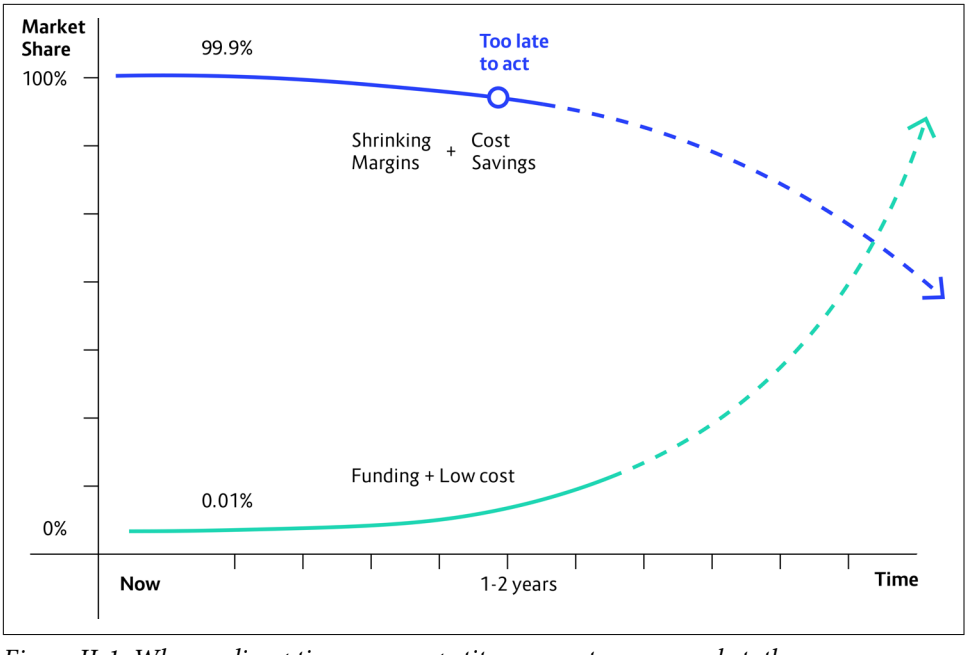


Figure II-1. When a disruptive new competitor comes to your market, they may seem small and nonthreatening at first—but their progress will be fast and relentless

Even if at first the stranger is just a small startup taking only a tiny bite of your market, pretty soon they are going to be eating your whole lunch! WealthGrid must take action before they even appear on the scene. And Jenny thinks she knows what to do.

Jenny has been to conferences, listened to talks, and read a lot of tech-industry articles. She understands that technology is the main factor in winning this competition. And that, furthermore, all of these technologies—public cloud, microservices, Kubernetes—are widely available to everyone. Many are even open source technologies, so they are available for free!

Really, this will just be a technical shift, she thinks. No need to put together a formal proposal and take it up the chain of command, because this falls into the category of pragmatic operations details that the C-suite avoids getting involved with. She can easily spare money out of her budget—have you seen how cheap it is to get started on the public cloud providers?!—and ask the engineers to work on implementing it alongside their usual tasks on the existing system.

This is how, without even going to her managers for approval, Jenny—together with the technical team she oversees—decides that they must go ahead and implement those things. Then they can show the entire company, once they have it ready. This should take a few months, no problem.

How Hard Can This Be?

The engineers are thrilled. Finally, a chance to roll up their sleeves and play with something really cool. They are eager to get started right away with their plan to put together a new platform for WealthGrid's software development system using Kubernetes, Docker containers, and microservices. Although they are accustomed to working with older technologies, the engineering team is confident they can figure things out as they go along. Jenny is confident, too—she knows her team is smart and capable. How hard can this be?

Their enthusiasm is only slightly dimmed when they realize that of course their first priority is the existing system—keeping it up and running well while still delivering new functionality. This, after all, is WealthGrid's core business, and where the company's profits come from. There are projects underway for building out new features, and those also need to keep moving forward. Even so, building a new cloud system shouldn't take too long, everyone agrees. They can work on it as a side project, when there is time, and still turn it around in a few months.

At first, things go pretty well. The team kicks off its cloud native transformation by setting up an account on a public cloud provider like Amazon Web Services or Google Cloud—all you need is a credit card, and you can be up and working almost instantly. If you want to run a single container, you can go online, click, and in 10 seconds you have a fully provisioned container running. It's practically magic!

Or perhaps they decide to go with a private cloud on their own infrastructure, à la VMware, and set up a Kubernetes cluster on a couple of spare servers … or get a free version of one of the commercial cloud native integration platforms like RedHat's OpenShift or Mesosphere DC/OS.

After that, setting up a small project is also pretty easy. The engineers play around with a single container, begin learning the ropes, and it goes really well. So then they try adding a small experimental app they whip together, or perhaps place an existing Java app exactly as is into a container and get that running on the cloud. Either way, at the beginning this initial trial project seems very easy—but quickly gets harder. A *lot* harder.

Dividing everything into related, containerized microservices makes intuitive sense, but no one at WealthGrid has ever worked with a distributed system before. The complexities inherent in even a single small and isolated project present a major learning curve, and there seem to be more and more things going wrong that they

just don't understand. Still, they are confident they can figure it all out, if they just find some time to focus.

Unfortunately, this dedicated time to focus on the new cloud platform simply never happens. Every time the engineering team starts to work on building out the new system, the work gets interrupted by a higher-priority feature request that comes from the market—from the customers out there using the existing WealthGrid products. The problem is, there is always pressure from customers. New or improved features are always winning the battle for assigning time to projects, and the new system always loses.

A Year Later, with Little to Show for It

So time passes, and after 12 months only a tiny portion of the new technologies is implemented. Even these are still strictly experimental. The engineers maybe have a few microservices running nicely on some Kubernetes clusters off on the side, but there is no significant impact on the current production environment. The results are intriguing—and also completely stalled. The team is constantly pulled away from any work on constructing the new platform by the need to continue rolling out features and support for the existing one.

This is not true failure, but there is no real success either. A year ago Jenny decided, along with her team, to go get for WealthGrid the same kind of responsive development and speed that their cloud native competitors enjoy. Their expectation was this would take maybe three to four months. Suddenly it's a year later, and there is little to show.

Jenny's first wakeup call was that WealthGrid needed to evolve into a cloud native company, and she responded using the resources at hand to start the transformation. Now she has a second wakeup call: what they're doing isn't working. It's not leading them fast enough to the goal of successfully competing with the disruptive newcomers crowding into WealthGrid's market.

Jenny needs to try a different approach. But what should she do?

CHAPTER 1

Tools for Understanding and Using Cloud Native Patterns

OK, so what just happened during WealthGrid's second attempt at a cloud native transformation? Why is WealthGrid having such a difficult time building a new cloud native system, even when they are putting tons of money and people into the project?

The short answer: cloud native is new, complex, and requires a very different way of thinking. Alas, this is apparent only once you have already moved a significant distance down the migration.

These eventual but inevitable difficulties take everyone by surprise because the first stages of a cloud native transformation are genuinely (and deceptively) easy. Engineers go to the conferences and they see Google Cloud Platform guru Kelsey Hightower or someone like that doing magic on the stage and producing results very quickly and efficiently. Then they go try it themselves by setting up one container, or maybe a simple application with a few containers running Java, and it goes great. It's actually pretty simple and can be done in just a few hours and they feel like they've got this, no problem-o. This is the point where most engineers go to management and say, "We have to adopt this technology! We love it—it's excellent and so easy we can do it without any big investment." So they get the approval, and move happily forward, expecting things to continue on being just that easy and successful—while they continue working the same way they've always done.

This kind of bottom-up push to go cloud native, originating from the engineers, is one of the two ways most enterprises enter into a migration.

The other type of push is top-down, where an enterprise's executives are hearing how hot cloud native tech is and decide that their company must do it too, even if they have no idea what it is all about. They are reading articles in professional publications

about how amazing cloud native is and hearing from vendors how easy it will be if they just spend money on the right products. Due to this lack of understanding, the tendency in the top-down scenario is to grab onto one concept—DevOps, Kubernetes, etc.—and order the tech staff to adopt it.

In both scenarios there is no actual adoption strategy, and the decision to move ahead is made based on lack of understanding coupled with the misbelief that the process will be fairly simple and straightforward. The problem is that the complexity of this is dramatically different as you start moving to a large-scale production environment. At some point you realize, *Uh oh, this is not going at all like we thought it would.* Figure 1-1 shows the cloud native adoption curve and how it moves from simplicity to complexity—and where the "Uh oh" moment inevitably occurs.

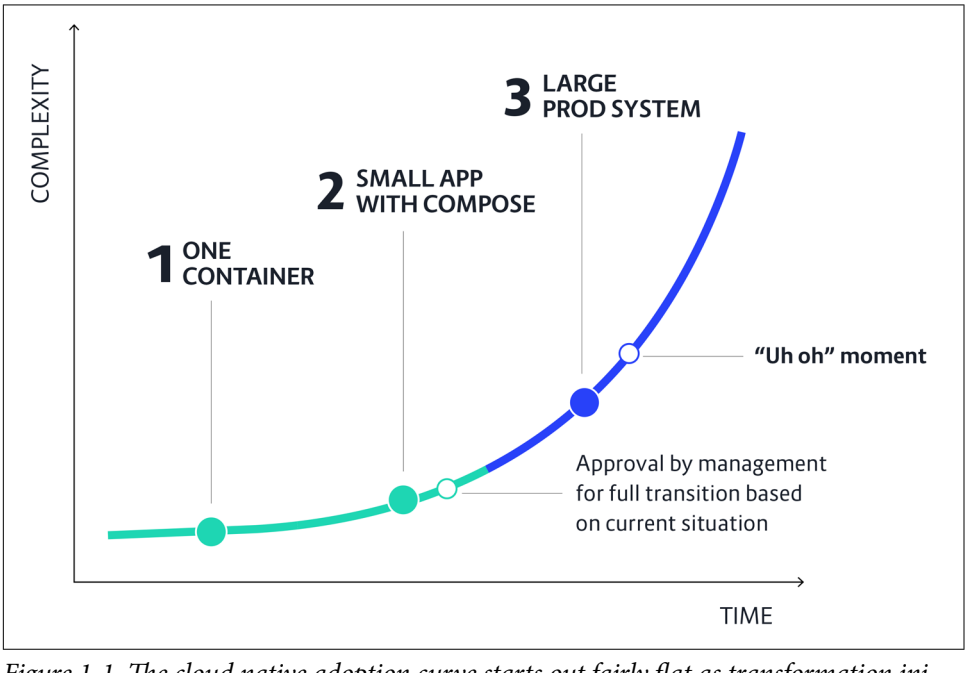


Figure 1-1. The cloud native adoption curve starts out fairly flat as transformation initiative starts out with early experiments with single containers and small test applications (using an external cloud database platform like Compose, to keep things simple). The curve gets steep fast, though, when it's time to turn these small experiments into an actual production system

This is exactly what happened at WealthGrid.

WealthGrid's first attempt to adopt cloud native was a classic bottom-up approach. Due to lack of resources, though, the initiative didn't get far enough for the team to reach the point where difficulties would begin to reveal themselves. Jenny's engineers

had had some success with building small pieces of the system, even if they had to start and stop a lot. Jenny's natural assumption, and theirs too, is that this seems actually pretty easy; the only thing holding us back is lack of time to focus on it. Based on their experience they fully believe that, if they just have more help and no distractions, they could crank out a new cloud native platform in just a few months.

So when Jenny first approached the CEO and board asking for support to go all in, company-wide, on a cloud migration initiative, this is what she told them. Naturally they approved her request and allotted significant resources. After all, top-down pressure had also been building at WealthGrid. The engineers just acted first.

For the second attempt the whole company tried going all in, sidelining the current system to focus on building a brand-new one as fast as possible. That didn't work either. In fact it went quite terribly wrong. What happened? And what else is there for them to do?!

So Many (Deceptively Simple) Tools

What happens? Why do things fall apart? The reason for this is best explained by the Cloud Native Computing Foundation's landscape—there are more than 1,200 projects currently monitored by the CNCF representing the cloud native ecosystem and its landscape. And there are new projects added every week!

Figure 1-2 shows the landscape at the time this book is being written, with its plethora of options. You don't have to become intimately familiar with all of them, but you do need to understand the basics of what they do, how they work, and how they fit together.

Gaining even the most basic grasp of this insanely large landscape means first understanding what the different tools do. Next, how all those tools fit together. Then, both how they integrate between themselves and how they integrate with older systems. Finally, you have to establish what kind of development and delivery processes you need to use in order to be successful using them all.

This requires a lot of new knowledge. It's not simply installing some new tech, "lifting and shifting" to the cloud. Not just updating the tools you use, but revolutionizing *how* you go about using them.

Let's introduce some tools. A few things that will help Jenny solve WealthGrid's problem.

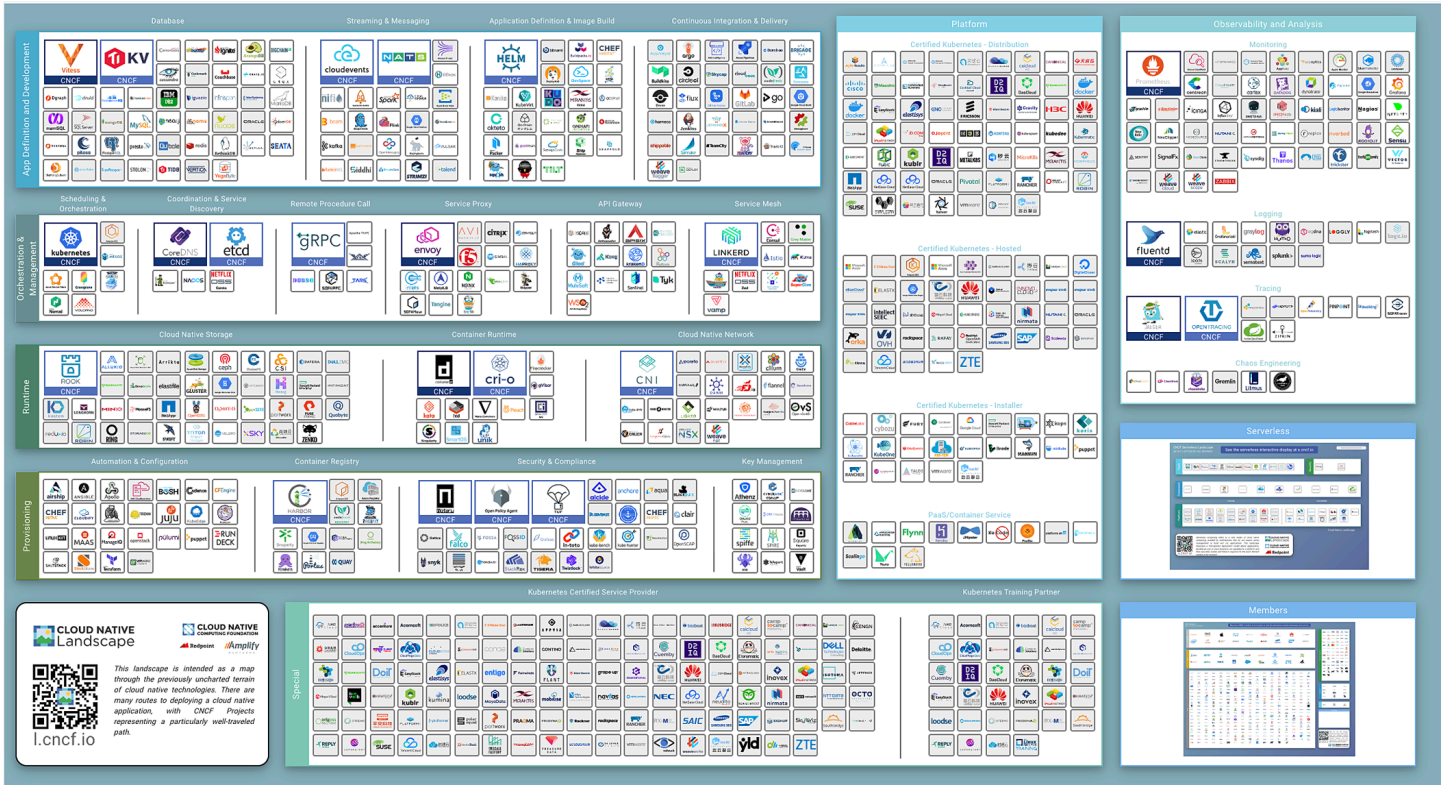


Figure 1-2. The Cloud Native Computing Foundation's cloud native Interactive Landscape (November 2019)

Tools to Take You the Rest of the Way

Before we can dive completely into the patterns, we first need to establish some of this new knowledge. The two concepts introduced in this chapter will provide crucial background understanding for applying the pattern design that follows. We are going to examine the important difference between proficiency and creativity and why managing them correctly is key to succeeding in any kind of digital transformation… like the one WealthGrid is attempting. Next, we work with McKinsey's Three Horizons model for visualizing how and when to apply creativity versus proficiency and, most important of all, how to balance them properly,

Proficiency Versus Creativity

WealthGrid is a highly proficient company. Things are optimized for delivering its core business mission consistently and well. This is a good thing, right? Stability, reliability, and quality drive profits and thus are valuable assets in any system.

Most traditional companies prize proficiency, the ability to complete well-defined and predictable tasks with maximum efficiency. This is how you deliver maximum value in a relatively stable context with few, if any, unknowns or surprises. In this context, creativity and innovation are viewed with skepticism. Innovation introduces unknowns into a highly regimented system—and with unknowns come risk.

However, as we have seen, companies in nearly every imaginable sector no longer have the luxury of a stable, predictable environment. Things are changing fast in every way, from new tech to new competitors. This calls for being able to respond—being able to change—to meet these new challenges whenever they arise. It means switching out of highly efficient, algorithmic delivery (proficiency) temporarily in order to change what needs changing—your product, your way of working, your company itself—through innovation and creativity. Once you've made the changes, then focus returns to proficient delivery of your improved products/service…until the next time a new challenge calls for a creative response.

So, yes, proficiency is a good thing…up until the moment that it isn't and you need to adapt to new circumstances. Most struggle to adjust, however, because they are trapped in proficiency mode. Again, many successful companies got that way through focusing on the bottom line: proficient delivery of their core product/service. They are really good at what they do, but in the process of becoming exactly that good at exactly that thing, they forgot how to work any other way.

Fortunately, once a company recognizes this reality it's possible to take corrective moves toward creativity. But which moves, and how?

Want Fries with That?

A useful tool for conceptualizing the flow between creativity and proficiency, which are essentially two sides of the same coin, is called the Knowledge Funnel, shown in Figure 1-3.

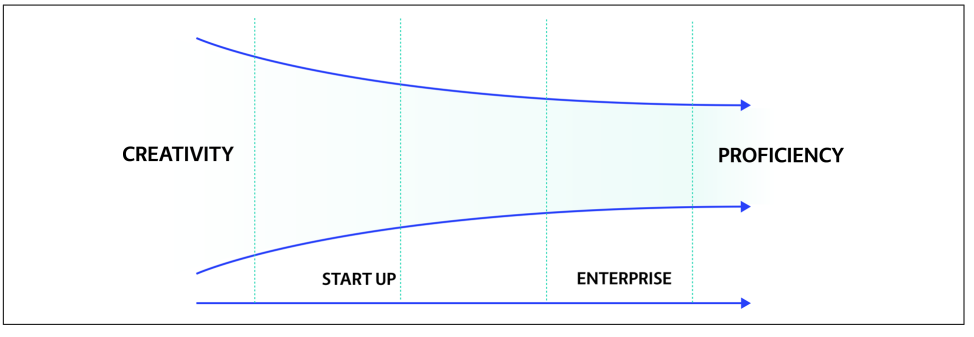


Figure 1-3. Roger Martin's Knowledge Funnel concept as applied to the process of moving from a highly creative startup to highly proficient enterprise delivery of products/services

The Knowledge Funnel was introduced by Roger Martin in his book *The Design of Business: Why Design Thinking is the Next Competitive Advantage*. According to Martin, every new idea goes through three major stages as it progresses to full adoption: Mystery, Heuristics, and Algorithmic. We are going to look at these stages in terms of McDonald's fast-food restaurants, because it is such an excellent example of pure proficiency at work.

Mystery comes first: you have a new idea and you think it's good, but you have no real understanding of how (or even if) it will work or how to implement it.

Imagine the very first McDonald's hamburger restaurant, founded in 1948 in San Bernardino, California, by brothers Richard and Maurice McDonald. It was a business serving food, but with one brilliant twist: you could walk up to the counter and get food right away instead of needing to sit down at a table and wait for service. No one knew at that point if this was a good idea or a bad idea; it was simply a new idea.

Heuristic follows the mystery stage. A heuristic is any approach to discovering something new or solving a problem that applies a practical method to reach the immediate next goals. A heuristic is not guaranteed to be optimal or perfect, only sufficient for getting things working well enough for the time being.

This new idea was apparently one whose time had come, because the first McDonald's restaurant was instantly successful. The McDonald brothers naturally thought to expand, opening new locations until they had four total. Things were still going very very well—Americans were eagerly embracing the concept of fast food—and they wanted to keep expanding. Starting with restaurant number five, however, they ran

into a problem. The quality of their hamburgers was not scaling along with the growth; it was going down, actually, because they couldn't keep things consistent. The McDonald brothers realized they had to keep production fast and quality high, but they didn't know how to scale.

The *Algorithmic* stage is the next stop on the Knowledge Funnel, representing fully optimized and streamlined proficiency.

Ray Kroc was a restaurant-supply salesman who was intrigued by the fact that the first McDonald's restaurant had ordered eight milkshake mixers. Most restaurants had one, at most two, and Kroc was curious to see what this new place was doing with so many more than the usual hamburger joint required. When he visited the place, he was instantly intrigued by their innovative approach. He was so impressed, in fact, that he persuaded the founders to let him try turning their concept into a nationwide chain of restaurants all based on a single, easily replicable model.

Kroc eventually bought the brand from the McDonald brothers and made McDonald's what it is today: essentially, a fully algorithmic operation. When a new franchise opens, it is done literally by the book: each franchisee is given a guide in which every step in running a McDonald's restaurant is well defined, extremely specific, and very clear. Every decision has already been made in order to optimize quality, consistency (a Big Mac in Birmingham, Alabama, tastes exactly like a Big Mac in Buenos Aires, Barcelona, or Beijing), and, above all, speed.

In short, the McDonald brothers came up with a truly innovative idea, but Kroc was the one who recognized what it could become—if proficiency were applied.

Creativity, Proficiency, and Enterprise

Unfortunately, when businesses evolve from a scrappy new startup to become a proficient and as-algorithmic-as-possible operation, they often forget how to be a startup —that is, how to re-enter the mystery state over and over in order to research and introduce new ideas to their business. This would be like McDonald's sticking to their original menu of regular burgers, fries, and shakes, never to invent their famous triple-patty Big Mac, much less Chicken McNuggets. McDonald's again is an excellent metaphor for our exploration of proficiency versus creativity, because the company has historically continued to introduce innovative new menu items (produced in, of course, a fully proficient fashion).

How do these concepts apply beyond fast food to the world of businesses and software, though? Figure 1-4 helps us visualize this.

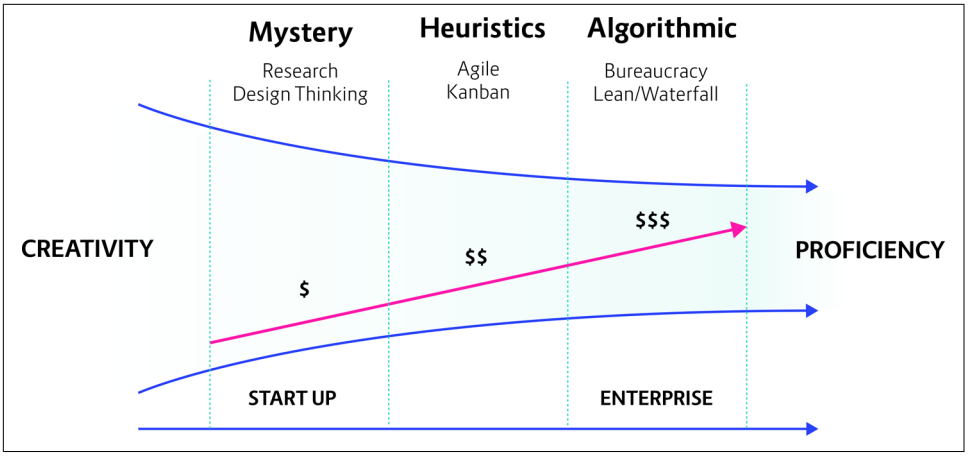


Figure 1-4. The flow between creativity and proficiency through the different stages of a company's journey from startup to enterprise, and from mystery to algorithmic delivery

In Figure 1-4 we still see the Mystery, Heuristics, and Algorithmic stages. Let's look at how they apply to typical enterprises.

Mystery: Startups by nature inhabit the creative phase—when building a new thing, by definition you don't know if it is going to succeed or not. The beginning phase, for a startup, is almost purely creative: the time when you have an idea and need to develop it. There are many, many ideas in the world, which is why there are so many startups. Most of these ideas are probably not very good, which is why so many start-ups fail.

Heuristics: Once your brilliant new idea has found success in your market, it's time to move to this next phase. In order to begin building up the company, to grow and find success in your market, you must become more proficient. This means turning focus from figuring out how to get your idea working at all (an inherently creative process) to how to get it working efficiently.

Algorithmic: You improve the idea until it becomes a viable business, and you can deliver reliably and steadily. Congrats, you have now reached full enterprise status!

Just as each of the three stages of business development has specific characteristics, how an organization functions internally while moving through each of them is also very different. There are whole books dedicated to exploring the mysteries of group behavior at different points along the path from crazy idea to fully established and delivered business value. Two of these in particular helped us crystallize our thinking around creativity, proficiency, and how to address and balance both through cloud native transformation patterns.

Loonshots: How to Nurture the Crazy Ideas That Win Wars, Cure Diseases, and Transform Industries, by Sahfi Bahcall, is a compelling look at how companies, individual teams, or really any group having a common mission will suddenly change from a culture of embracing wild new ideas (while in Mystery mode) to rigidly rejecting them (once Algorithmic stage is reached), just as flowing water can transform into inflexible ice. Bahcall draws upon the scientific field of phase transition to illustrate how group behavior is influenced by the structure of its containing organization.

Likewise, Daniel Coyle's *The Culture Code: The Secrets of Highly Successful Groups* examines how diverse groups end up internalizing the same homogenous mindset and how this can lead to the stifling of innovation within an organization.

What do these look like over the life cycle of a typical enterprise? In the beginning you want to use research, design thinking, all kinds of creative ways of doing things, because this time is all about exploration. As a successful direction emerges, you begin to define your product or service and your process for delivering it. However, you are still able to pivot quickly and change direction because there is still some flexibility and creativity in your processes. If things turn out not to be working, it's still possible to change course.

At some point, though, when you have honed things down to delivering a core value proposition, you reach the algorithmic stage. The inevitable outcome of this is building in bureaucracy to keep things running smoothly and consistently. This happens because it is only by reaching the algorithmic stage and full proficiency that you can generate significant revenue. Simply by being successful, you undergo a process of organizational evolution. Early on, you are a handful of crazy people trying all kinds of wild things, and the point is not to make money (well, not yet, anyway) but grow your idea. Once you prove your idea works, though, you want to grow the business.

In such cases, companies tend to become Waterfall organizations with a strong hierarchy, or perhaps a Lean one like the Toyota Production System. The Lean production model focuses only on that which adds value, mainly through reducing everything else (because it is not adding value). Unfortunately, this almost always

means jettisoning creativity, which has no functionality in a proficient system because you no longer have an idea you need to explore. Now you have a successful product and the sole need becomes only to keep delivering that product as cheaply as possible. Perhaps that product improves over time, but the focus is now all on proficiency and no longer on evolution.

This does not mean proficiency is inherently bad or creativity is automatically good. An enterprise needs both, in proper balance—a balance that can shift according to need. There always needs to be at least a small portion of effort going to improving and trying new things. Because if you are only proficient and some kind of disruption or other significant change happens unexpectedly—one that requires you to shift to a new or different way of doing business—you have no way to respond.

When an existing business gets challenged by the stranger in their town, they swing from proficiency to overvaluing creativity, because they suddenly realize they don't have enough of it. Startups, meanwhile, tend to overvalue proficiency, which is what *they* don't have since they are often seat-of-the-pants organizations with few resources. They aspire to the stability that proficiency represents, but this leads to a different problem. Many startups jump too soon in trying to introduce quality and so try to establish an algorithmic delivery model too early, locking themselves in and diverting resources when they still need to be focused on fully developing their idea. Thus, they are trying to leave creativity prematurely in favor of proficient delivery of a product that is not yet fully baked.

So, yes, you need both proficiency *and* creativity. One is not better than the other, or more important than the other—it's the balance that is important. You need both, but not at the same time. This is because they both need to be managed differently. You cannot have teams that are working both proficiently and creatively at the same time, since these are conflicting mandates:

- Proficient teams require high repetition to deliver the same thing, over and over, very efficiently and reliably, and at the highest quality possible. High repetition, high feedback, small set of very specific rules. The emphasis is on skills and repetition.
- Creative teams, on the other hand, have no specific list of tasks. Their work requires open-ended thinking that is more like puzzle solving. This doesn't mean that creativity equals chaos: there is still a guiding purpose behind it, and tools to use. To effectively nurture innovation there must be a goal and the strong support and safety of a space that allows open-ended experimentation. Autonomy is crucial: once the goal is established, let the team find solutions in whatever way they can discover.
- Both types of teams are just teams, composed however your organization's team structure works. It's their jobs that are different: the proficient teams are your

bottom-line delivery workers, the creative teams are focused on research and next steps. Typically you're going to have more teams tasked with delivery than innovation/creativity.

Trying to have one team work using both proficient and creative approaches at the same time is simply futile. They are two different mandates, and they conflict with each other. You are asking a single team to focus on optimizing repetitive ongoing delivery processes while simultaneously innovating on them, which means neither area is going to get full attention or best effort, and both are going to suffer.

Thus an organization needs both kinds of teams, and they need to be distinct and separate.

They also, of course, need to work together. The proficient teams focused on bottom-line delivery of product need to tell the creative teams—who are in charge of keeping the company looking forward and engaged with innovation—what the actual problems are in the market that the company needs to solve. The creative teams, meanwhile, need to return something from their development pipelines that the proficient teams can use in real life that is useful to customers.

Ideally this is a dynamic relationship—a well-adapted organization can move as needed between times of greater proficiency versus times of increased creativity.

It is difficult to achieve this, but possible. Striking the balance requires maintaining the separation of proficient and creative teams while closely coordinating between them. Different styles of management are required for each, and a designated champion (or two, or even more) is needed to act as a kind of translator to manage whenever there are handoffs between their respective efforts.

The way to integrate proficiency and creativity, organizationally speaking, uses our next tool: McKinsey's Three Horizons model.

Three Horizons

Once you understand the differences between creativity and proficiency, and the relationship between them, we use the Three Horizons model to understand how to blend and balance investment in developing new products within a company, while still delivering efficiently and reliably (and, of course, profitably).

The Three Horizons is an incredibly useful tool for identifying your business's current state and then strategizing adaptive actions—either to take now or to hold for when they might be needed in the future. In order to use it most accurately, we break creativity into two different categories: innovation and research (more about those below). Let's jump right in: Figure 1-5 shows the three stages.

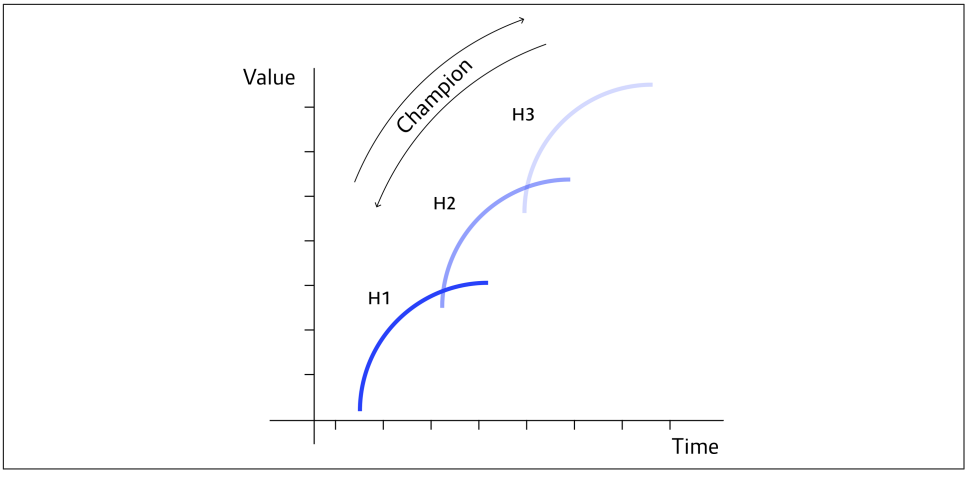


Figure 1-5. McKinsey's Three Horizons Model showing the relationships between delivery, innovation, and research

- **H1:** Horizon 1 represents your current core business—presumably, the things that provide your cash flow and main profits. This also includes logical next-step development of/iteration upon any product or service you are making right now.
- **H2:** Horizon 2 is investment in innovation: taking ideas that have been shown to work in the H3 incubator and productizing them for real customer use. This means both introducing new technologies and new ideas into existing products or changing your existing products themselves to something entirely new. At this point you are working with a promising idea that looks like it is going to grow to become a successful part of your bottom-line business. (Interestingly, cloud native itself is now still mostly at H2: it's working at many organizations but is far from easy or even completely stable and requires special knowledge to implement.)
- **H3:** Horizon 3 is research. Pure exploration of new ideas, research projects, pilot programs. Nothing that you can really use right now—here is where you are looking at the long term, things you may need to know or use a few years down the line. H3 is about awareness: staying abreast of what is coming next so you at least have some understanding when it does get here. Some of what you discover in H3 will get moved to H2 for further investment and development. Not everything you investigate in H3 will pan out or end up being useful, but that's OK. Sometimes experiments just don't work.
- **Time:** The x-axis does not represent a linear progression of time; it is not telling you when to apply each stage—now, later, or far in the future. *Companies must work within all three horizons at the same time.* In this model, "time" demon-

strates how an enterprise moves, throughout the course of a normal venture's life cycle, between the three interrelated cycles.

- **Value:** The y-axis represents the growth in value that organically occurs when an organization addresses all three horizons simultaneously and, as we shall see, in proper balance.

Companies start with some crazy big idea and founders who create a company around it. If you find success, you have to hire more people, which means starting to create specific heuristics so everyone can work together. If you are really successful, at some inevitable point you form a hierarchy, with set positions, routines, processes—it's a one-way ratchet. Nobody wants to go back to the zero-process chaos of an early startup, no matter how creative a time it was.

What this means, though, is that most enterprises in the world use economy of scale to optimize both operations and profits; naturally, then, it becomes where all their efforts focus. Only a very few companies keep creativity on the side and keep it feeding in.

An adaptive business constantly evaluates and recalibrates the relationship between its three horizons, pursuing the optimal balance between proficiency and creativity. To manage this across an organization, it's important to have people called "champions," who understand those different horizons and move the technology across those three horizons.

We Are the Champions?

The champion is the person who keeps a firm fix on the bottom line while also pushing the likely next step—and keeping an eye on whatever crazy future thing could be coming next. It's a lot to track, and so it's good to have someone officially in charge of doing just that.

Champions are well aware that for a good company in a stable market, a good general ratio between the three horizons means putting most of the company's efforts into delivering the core business. A small, but still significant, portion gets directed into practical innovation: basically, building the logical, market-demanded next step in terms of feature or functionality. Finally, a little bit gets left open for research.

Of course, this ratio represents a "business as usual" situation. It should change whenever the business environment shifts, in direct response to the type of change—like when a disruptive stranger suddenly shows up in your market.

Companies who lack a champion (or, more to the point, aren't even aware of what a champion does and why they should have one) often skew the balance. This happens unintentionally, because they are fully focused on the business at hand and forgetting to build in some innovation to keep their creative juices flowing.

Figure 1-6 puts some numbers behind these varying scenarios and the proper ratio for each of the horizons, in different circumstances. Note that these are approximate numbers to demonstrate the relationship between proficiency, innovation, and research, not hard-and-fast definitions that you must hit exactly (or else). They are based on our own experience with what we have observed works well in companies that achieved not just a successful cloud native transformation but also a new ability to respond to changing circumstances as they arrive.

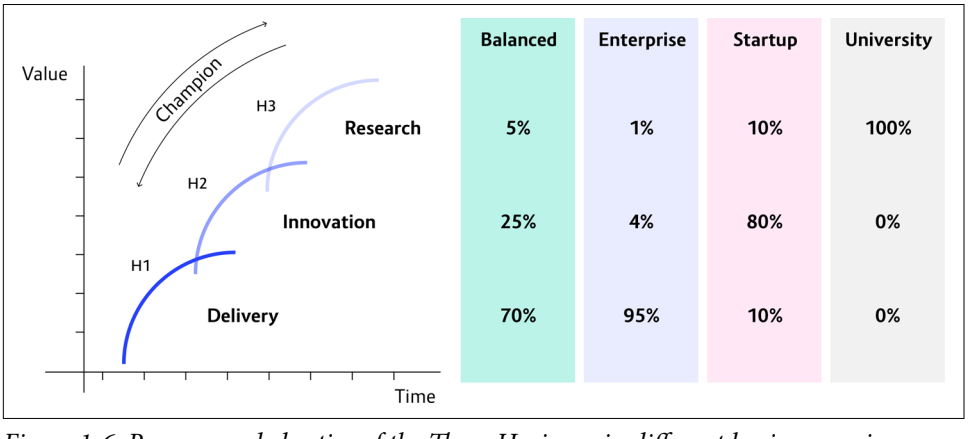


Figure 1-6. Recommended ratios of the Three Horizons in different business environments

- **Balanced:** Normally you would want to invest 70%–80% of effort into delivery, invest 15%–25% into innovation, and maintain a small budget for research (5%).
- **Enterprise/Full Proficiency Focus:** Many enterprises, though—the ones we mentioned earlier, who forget how to be creative in the pursuit of pure proficiency—look more like 95/4/1. To be honest, this is even more realistically 95/5/0 at most traditional Waterfall companies. Too many focus solely on delivery, caring only about the products they are selling right now.
- **Startups/Mostly Innovation Focus:** By nature, startups are an innovative type of company. They are in the process of building a proper product, so they mostly live in the middle. They may dabble a bit in H3 (10%), and of course they are working their way as fast as they can to inhabit H1, so there is some groundwork already laid there (10%). But H2 is their home turf (80%).
- **Research:** Universities are pretty much the only entities you will find spending 100% of their time and effort on research. These are the only places that can do pure exploration without needing to worry about monetization in the near future.

Dedicating 5% of a company's resources to research may not sound like a lot, but it's crucial. That 5% is where you are gaining knowledge and retaining your ability to be creative when necessary. Where this goes wrong, where companies end up in trouble, is when they are not thinking—there is no strategy—and they try to move straight from research to delivery.

For example, in a cloud native transition, a company's engineers might try to adopt a microservices architecture when they know nothing about it, have no background, and so end up approaching it all wrong. They researched enough about microservices to recognize that this is the right thing to do, but they are rushing to get things working, which means they skip the middle stage and try to squeeze it directly into delivery. Skipping H2, which is the pragmatic development phase for creating heuristics around delivering new ideas, might sound like a way to speed things up. But instead it's a recipe for failure, since they haven't taken time to understand how it works and fits together. An innovation champion's job is to prevent just this sort of short-sighted corner cutting.

Now let's look at the flip side. Sometimes, proficiency is, for the most part, what a company needs, with side initiatives into innovation and pure research. And these can even go to zero, temporarily, if needed. An example would be when quality is suffering in an enterprise's main offering, and focus needs to be fully on delivery for a while. But this is a temporary pause, and in a healthy company (especially one with a champion on the job) once the crisis passes, the ratio gets rebalanced.

There is another scenario, however, where many companies do exactly this—pull all resources out of innovation and research in order to focus on the bottom line—and it is absolutely the wrong move. This would be when a company is under serious pressure from a new competitor, or a previous competitor that has introduced something new that is disrupting the whole market. Suddenly the established company is rapidly losing market share, and its leaders panic. The most likely path for a traditional company in this case would be to start cutting costs, especially in development. In times of distress, the R&D department is the first to suffer. But by doing this, the company is basically guaranteeing that it will not ever be able to adapt to the new market conditions and emerge as an equally strong contender.

So What the Heck Happened at WealthGrid?

How do we use these tools to understand what went wrong—and make it go right?

When we first met WealthGrid, it was a classic Waterfall-process company focused on the bottom line: delivering its core business value as efficiently and profitably as possible. If WealthGrid had an innovation champion, which they most assuredly did not, this person would have analyzed the Three Horizon ratio as 95% of efforts invested in proficiency, 5% in innovation, and 0% in research.

It was in this 95/5/0 context that Jenny and her team first attempted to move WealthGrid onto the cloud. As we have seen, this did not work. At the time, they chalked it up to not having enough time to focus on the migration because the existing system always took priority. We now have new tools to gain a deeper understanding into the broader forces that caused this to happen.

Essentially, WealthGrid's first attempt failed because they tried to implement cloud native—which requires a creative and innovative approach—from a proficiency mindset.

Jenny and the engineers put cloud native technologies and tools into the Scrum backlog along with all their other tasks for running the existing proficient system. Then they tried to execute it using the same type of processes they've always used, using the same sort of management by deadlines, sprints, and stress. Many companies use sprints as a development framework; Scrum usually sets a goal with two weeks to deliver it. The problem is that when you are running as fast as you can to deliver, it's hard to look around you with a mind open to innovation: it is impossible to be predictively creative.

So this initial attempt did not work. The dirty secret here is that, even had Jenny's team been able to work 100% only on building the new system, they still would have failed. And for the same reasons, though perhaps from a different direction, that attempt number two also then proceeded to fail.

WealthGrid's second attempt to go cloud native was basically the full reverse: *Now we will go into complete creativity!* The company moved a majority of resources—in this case, people—to the transformation project. That is, they redeployed most of their engineers from working on the original proficient system to building the new one. Dedicating resources to a transformation is a great first step. Unfortunately WealthGrid then took multiple wrong steps, all of which added up to the same thing: still attempting to deliver this brand-new technology and way of working by using proficiency-centered processes and culture.

There was no adoption of new ways of thinking or doing things, no application of a design thinking process to identify new ways to do creative problem solving, no pur-

pose setting. The second attempt to build a cloud native system still used a Scrum and/or Waterfall approach to building. Just about every engineer in the company may have been playing around with different tools and cloud providers, but they were all using the same old knowledge and the same old way they have been doing successfully (and proficiently) for so long. This is why things went wrong for WealthGrid—*both* times.

Cloud native is a new way of doing things. It is not predictable, at least not to people who lack a good understanding of how it works. Most of the people at WealthGrid did not have this knowledge. Probably no one, not even Jenny, truly understood the full intricacies of cloud native architecture; certainly, no one at WealthGrid had any experience actually building a cloud native system. Lacking this understanding and experience, they of course used what they knew, the tools and techniques at hand. They didn't know what they didn't know.

The initiative may have moved forward pretty well at first, but inevitably the complexities multiplied as the system grew in size. Things slowed down more and more until the entire project simply could no longer progress at all. Meanwhile, the old system—the one WealthGrid is still running on!—had been languishing, with no new features or functionalities being delivered to the customer for over a year.

This was the third crisis point. WealthGrid was still committed to becoming a fully cloud native company. But it also needed a way to continue delivering value to customers while it worked to find the right path—the middle path between proficiency and creativity—to finally deliver the long-delayed new platform.

Summary

Digital transformation, ultimately, requires a balance between innovation and pragmatism, between creativity and proficiency. Some companies attempt to innovate but do so by trying to deliver creativity using proficient processes—that is, long-held practices and beliefs that worked well for them historically but don't work with cloud native architecture. This leads to failure, or at best a low-functioning, improperly implemented attempt. There may be a few microservices running on a few Kubernetes clusters, but no real value is being delivered—especially in comparison with how much time and money has just been invested.

Others go all in on innovation, attempt to abandon the old system completely to build a new one from scratch, and still get lost. Many times these companies are trying to be like Google, one of the most creative (not to mention fully cloud native) organizations around. The common misbelief is that being like Google means being all in on creativity—let's say something like 98% creative. Google's real focus, however, is very much on proficiently delivering their existing products and services while investing very intentionally in small but targeted and highly impactful creative

initiatives. Putting in the terms we have been using in this chapter, Google's balance would actually be closer to 2% creativity and 98% delivery. The point is, they do have a balance, and it is what works for them. The real problem for most established and successful companies is that they have no idea how to be creative at all, in an effective way, at any number.

Proficiency is important. Creativity is important. Neither is better, and both are necessary. Proficient teams need to be managed in a way that supports their focused, stable and efficient delivery of bottom-line core business value for the company. Creative teams are managed for open-ended exploration of next steps, so the company stays innovative and ready to take responsive and adaptive next steps whenever needed.

It's not surprising at all that WealthGrid attempted to transform itself into a creative cloud native company using proficiency-focused Waterfall/Agile processes. This is an extremely common thing, and we have seen it many times, in enterprises of all sizes and from all sectors. Certainly, if they knew better, they would do things a different way. The right way.

Now let's see what the right way looks like.

CHAPTER 2

Applying the Patterns: A Transformation Design Story, Part 1

This is a detailed design for a cloud native transformation. Here we will lay out patterns from start to finish, explaining the order and reason for the choices as we go along. It's a lengthy and involved process, so we have divided it into two chapters. This one covers the period from pre-initiative prep through the point where we have researched, experimented, and prototyped until successfully uncovering the likely best transformation path. Part 2 picks up with verifying the path and beginning to build a production-ready cloud native platform, carries through onboarding everyone onto the new system with a new way of working, and then moves to shutting down the old one.

For several chapters now we have watched WealthGrid struggle with multiple attempts at a cloud native transformation, only to fail each time. The story we are going to tell now is how to do it right—in other words, what WealthGrid would have done, had they known better. We will be using patterns to show the way, so that now *you* will know better.

First, let's take a quick review of the story thus far.

WealthGrid's first erroneous attempt to move to cloud native is an extremely common strategy: treating the transformation as only a minor technology shift. Many companies try exactly this, and equally many fail at it. The technology is new and rests upon a complex distributed architecture, so right away you have two things nobody at WealthGrid (or pretty much anybody at any other company, aside from a handful of tech giants) really understands or has any experience with. Setting a small team with no background in the tech to deliver a full transformation as a side project while also doing their usual work on the existing system? We saw how well that worked for WealthGrid—or, rather, didn't work.

WealthGrid's people, Jenny and Steve, as well as others, saw it too. And they did try to correct course. For their second attempt they tried going all in, assigning a big team (and big budget) to bringing the whole organization onto the cloud. Unfortunately, this all-hands-on-deck approach did not work any better. First, too many teams tried too many experiments and came up with too many possible solutions, none of which fitted together. Six months into the initiative—their original deadline—they were no closer to having a working cloud native platform than they had been at the start.

Again, they tried to adjust. They tried to clean up the too-many-platforms mess by calling in a systems architect (with zero experience in Kubernetes) to design a unified approach. He came up with an impressive diagram that, unfortunately, got it all wrong. Even more unfortunately, there was still not enough understanding of how cloud native works, within WealthGrid's ranks—from corner office to middle management to engineers—for anyone to see how wrong it was. Thus everyone set to work trying to implement the new but still nonfunctional architecture. Again, everyone worked hard. Six more months passed. Again, nothing was delivered.

Meanwhile, the skeleton crew of engineers maintaining the original system was able to keep everything working, but they weren't able to deliver new features. The team was understaffed. And besides, why invest in new functionality on the old system when it was about to be replaced? Initially nobody worried about this, because the new cloud native system would have such velocity that WealthGrid's feature debt would surely be paid back in very short time. But then the promised six-month delivery time turned into a year, and still the new platform was nowhere near ready.

This meant a year of no new features or other meaningful improvements for WealthGrid's customers—and a very real risk of losing market share. The sales and marketing teams got the ear of the CFO, who talked to the CEO and the board, who finally laid down an ultimatum: Here are five new features that we need, and fast. We don't care about which platform we use. *Just deliver them*.

Now what does WealthGrid do?

PHASE 1: THINK

Phase 1 of the transformation design is called "Think," because it's all about ideas, strategy, and objectives.

Most companies are going to enter this phase at a very similar standpoint: they are using Waterfall to Agile-ish delivery methods focused on very high proficiency. This means that most of the talent and resources in the company are dedicated to delivering a very stable core business product or service. There might be a bit of experimenting going on here and there, but it is definitely not an internal priority. There's probably no team dedicated to innovation, or any programs designed to inject crea-

tivity into the existing system. In short, this is very likely a company that has forgotten how to be creative.

To put some numbers on this, at this point the distribution of investment between delivery, innovation, and research is 95/5/0. The vast majority of focus, 95%, is on delivering the core product as efficiently as possible. A small amount, 5%, is left over for any innovation going on. Whatever this is, it's targeted mainly at improving systems around the core product, and expected to be useful or pay off very soon. Zero research is going into any kind of crazy ideas that might—or might not—pay off big someday.

It is against this backdrop that the first stirrings toward transformation take place. Our first set of patterns addresses getting the process started: first, effective strategic thinking and decision making from the organization's leaders. And then converting this strategy into vision and objectives to move the process to the next phase, execution.

Enter the Champion

Any significant innovation always begins in the same place, or rather with the same person: the Transformation Champion.

A large, established system in motion is usually slow to change direction: see Newton's First Law of Physics, also known as the law of inertia. And WealthGrid, like any other sizable and successful company, has invested a lot of time—years, if not decades—in streamlining its core business processes to be as proficient as possible.

The problem with achieving this impressively high proficiency tends to be a corresponding decline in the ability to innovate. Every successful company was once a startup, and startups are all about creativity. Its ultimate goal, though, is to get to proficient, even algorithmic, delivery of their product or service, because that is where the profits lie. Once they make it, successful enterprises typically focus almost exclusively on operating as lean as possible. This is fine, good even, except that when they focus entirely on proficiency, they forget to keep a piece of creativity alive.

So when any company similar to WealthGrid decides to change course—i.e., innovate—even when it's in response to an undeniable threat to its bottom line, it must overcome a fair amount of resistance. Just as a catalyst like a disruptive stranger coming to your town is needed to stimulate the change, a corresponding force—a metaphorical town mayor—is needed to manage the community's response. In a cloud native transformation, this person is the Transformation Champion and the whole thing

starts with them. There has to be trust in their leadership and willingness to follow their guidance.

PATTERN: TRANSFORMATION CHAMPION

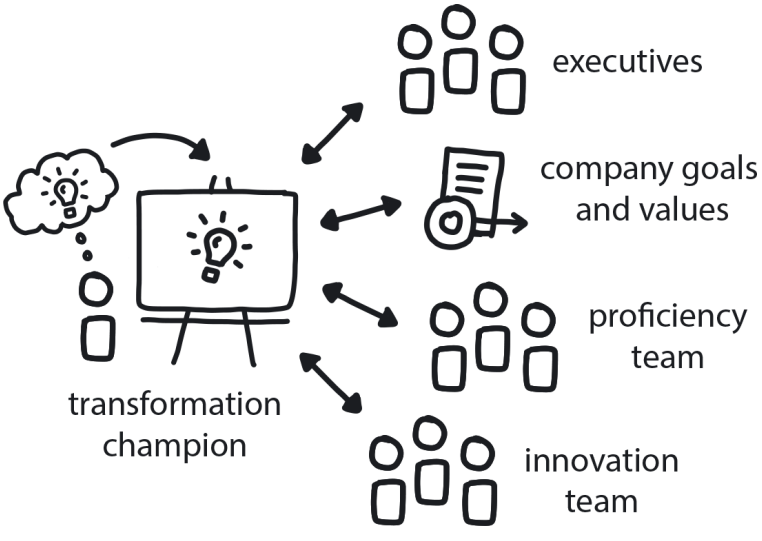


Figure 2-1. Transformation Champion

The Transformation Champion is a person (or sometimes a small team) who understands both the transformation and the company objectives, is well connected within the organization, and is highly motivated to promote the transformation. Unless they are recognized by the company's leaders and given authority, however, they will be unable to stimulate effective change across the organization.

Every company has some of these people. They see the future more clearly than most, and as a result they always want to change something. This is the hallmark of a true champion: they don't just have ideas; they try to take action. Champions are good at promoting a new idea while making sure it is a proper fit for the company's goals and values. Such people need to be identified, nurtured, and trained in general, but when it comes time to undertake a major initiative that will terraform the entire organization, they are critical for its success.

The trick is recognizing the champions in your midst, because this first pattern is not about *creating* a Transformation Champion—it's about *discovering* the one(s) you already have. These people tend to be self-selecting, and quite often they are the trigger for the whole thing.

So, look around! In every company we visit to consult about cloud native, there is one person super excited we are there. This is often the person who asked us to come in the first place.

Once we take on the client, our first question is, will that person lead the initiative, or will they name a different leader? This can be tricky. Not everyone is cut out to manage a large transition project. But in general this self-selected evangelist is going to be the right person to lead yours, and so it's important to appoint someone else only if there is a truly compelling reason to do so. (Said compelling reasons, FYI, do not include "In our company senior managers always lead large projects." Trust us on this. You're going cloud native now and it's time to do things differently.)

If you seek your champion, you will find them. Somebody inspired this whole transformation thing, and this person could be anyone anywhere in the company, engineer or manager or CEO. Finding this person is not the problem—they exist. The true problem is that, usually, they just don't have the power to get things done right. Fortunately, the simple fix is simply to empower them and then get out of the way!

So, once you find them and anoint them as the authorized Transition Champion, let this be known far and wide. Send a clear message that this transformation is an initiative supported by the company—not just a side project by a few motivated employees—and that participation is not only appreciated but expected. This is absolutely crucial. A champion can tell everyone the absolutely right thing to do as their part of the migration, but unless they are fully supported by the executive team (or are themselves very senior) people can ignore them and do nothing, or even block their efforts.

WealthGrid's Transformation Champion is Jenny, by the way. She is a classic example: Jenny saw what her company needed to do in order to be ready for the ways the market is permanently changing. And then she stepped up to make it happen.

You may rightfully wonder why, if Transformation Champion is an essential pattern and Jenny is a good example, did WealthGrid's efforts then fail not once but three times? After all, Jenny knew what needed to happen: the company had to go cloud native in order to remain competitive. She even, eventually, had the full confidence and backing of the CEO and board to do a full-on initiative, not to mention hands-on help from much of the company's engineering staff. What was missing?

The missing piece was Dynamic Strategy.

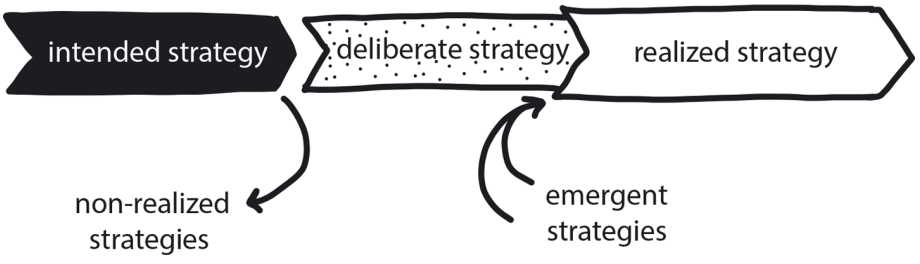


Figure 2-2. Dynamic Strategy

Today's tech-driven marketplace is a constantly shifting environment, no matter what business you are in—so your game plan needs to shift right along with it.

WealthGrid's first transformation attempt, with Jenny's team trying to do it as a side project, did not have any real strategy at all. In the second attempt, though, everyone got on board, a strategy was created, and things still went sideways. Had the Dynamic Strategy pattern been applied, however, the story would have made the same missteps, only much more quickly. Instead of wasting a year to figure that things were simply not working, they could have moved through the process in two months. It would still have been painful, but at least it would have been quick.

Dynamic Strategy is essential for doing cloud native right, from inception to completion. This is actually a super pattern, one so essential that it overlays all the other patterns in our transformation design from the very beginning, and stays forever. Let's look at the next set of patterns now to learn more specific steps for getting a transformation off to the right start.

Ready to Commit

Even though it's time for enterprises to learn how to navigate this new paradigm, we aren't saying to forget the past. We learned a lot from WealthGrid's experience thus far, and we need to use it. The first lesson being, don't continue something from the past just because your organization started it. That is classic sunk cost fallacy, or irrational escalation: the phenomenon where people justify increased investment in a decision, based on the cumulative prior investment, despite new evidence suggesting that the decision was probably wrong. This unfortunately is a very frequent phenomenon in companies moving to cloud native. People routinely push forward with

projects that are obviously not going to bring any value, just because they've already invested so much in the mess.

So let's use Dynamic Strategy to pause and reevaluate here at the very start, and make sure we are not going into sunk cost. Now is the time to ask, *Do we even need this thing?* using the Business Case pattern.

PATTERN: BUSINESS CASE

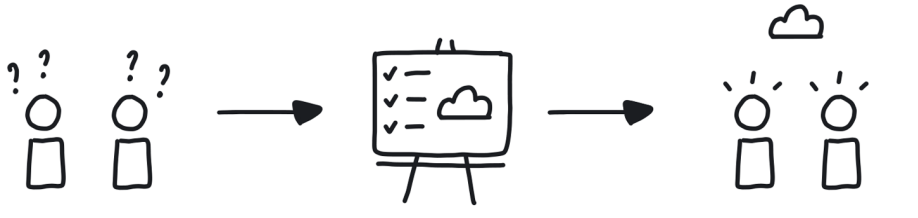


Figure 2-3. Business Case

When an organization's executive leadership does not fully comprehend the advantages that result from a cloud native migration, a strong Business Case allows them to understand and support the project without hesitation.

Sometimes Business Case can be a tough sell even when a cloud native transformation is clearly the right response to an existential threat. The traditional model is for organizations to be massively risk averse, which means minimizing uncertainty at all costs. This is why it takes forever to plan anything in Waterfall—the process is designed to identify and mitigate every conceivable problem in advance. So pitching a cloud migration to a change-averse culture that avoids new technologies or experimental approaches can be challenging. Cloud native is complex, and the benefits are not easily visible, especially at first. So, for all these reasons, some companies are simply going to avoid talking about the risks and advantages. In these situations, Business Case facilitates the conversation in a factual, objective, and non-threatening way.

For most, though, Business Case is a valuable reality check before diving into an initiative they are already inclined to take on. Too many organizations jump on the cloud native bandwagon without doing a true cost/benefit analysis. (In fact, the cognitive bias known as the "bandwagon effect" exactly describes how companies caught in the hype of the cloud conversation make decisions without understanding exactly how a transformation fits with their business needs and goals.) Thus, evaluating the Business Case should involve business stakeholders, information-gathering inter-

views with internal and external subject matter experts, and a clear vision for where the organization is headed.

This may reveal that going cloud native is not actually the right thing for a company to do, at least not right now. There are a few times when cloud native likely would not be the right move, and an organization working through Business Case needs to be aware of them even if they're uncommon.

- When a company is struggling for survival and has no budget or time to allocate to the transformation.
- If there is an immediate crisis that cannot be solved with cloud native. This is a case of "right move, just not now." Cloud native will have no appropriate priority in this case, but once the situation has resolved, Dynamic Strategy tells us to revisit the decision.
- Maybe in a very stable market where nothing changes. But these markets are very few, and becoming fewer all the time, so this is a high-risk decision.

The best reason not to do cloud native at all is when the software part of the business is really simple and limited, and cloud native will overcomplicate things without adding significant value. But this mainly applies to smaller companies, since any large enterprise should have substantial IT that will benefit from a migration.

Everyone else should start investing in cloud native. The only question is pace: some can do it slowly, while others—for example, anyone in a stranger-danger situation—should do it with higher urgency.

Keep in mind the difference between cloud computing and cloud native. Namely, the cloud refers to on-demand delivery of virtual infrastructure (hardware/servers), storage, databases, etc., while cloud native is an architectural approach—a methodology. Even monoliths can benefit from taking advantage of the automation, ability to pay only for resources consumed, and other niceties built into public cloud platforms. Any enterprise running a highly stable monolith on a legacy codebase that requires very little change or updating ever needs to take a very hard look at the Business Case for regrooming that monolith into microservices. (Don't stop reading, though. There are many more patterns to help you move your functional monolith onto the cloud!)

Business Case in hand, the next step in our transformation design is the Executive Commitment pattern.

PATTERN: EXECUTIVE COMMITMENT

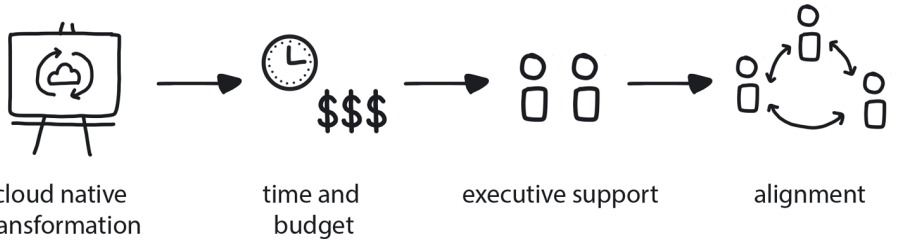


Figure 2-4. Executive Commitment

To ensure allocation of sufficient resources and reasonable delivery time frames, large-scale projects such as a cloud native transformation require strong Executive Commitment.

This function of this pattern is to establish the cloud native transformation as a high-priority strategic initiative with explicit support from the company's executive management. Public announcement of the cloud native transformation as a key strategic initiative creates company-wide alignment and awareness, while also setting the expectation of collaboration from all departments within the organization.

It might seem that Executive Commitment for a transformation should go without saying, but it's an important moment in a migration. It establishes that this is not some small engineering project driven by a handful of motivated employees but a full-company effort, and the senior leaders are standing behind it. This is needed for a number of reasons. First and most obvious is that commitment from the company's most senior leaders makes sure the project gets an appropriate level of resources and budget allocated. Less obvious but equally important is that announcing their full support makes it known across the organization that change is coming, and that the cloud native transformation is an official part of the company's value hierarchy. Maybe the project doesn't need to be *the* top priority, but it is indeed a real one.

WealthGrid's first attempt failed in part due to a lack of Executive Commitment. Though Jenny got permission from upper management, she didn't get support. No extra budget or resources were given to the project, and it remained a small, localized effort that most people in the company had no idea was even happening. This is exactly the reason why it all gets spelled out as a pattern: Such commitment from the management needs to include preparation of a Transformation Strategy (pattern coming up), public announcement of the project, and the allocation of adequate

resources and budget. Again, had Jenny gotten full Executive Commitment for her first try, it would have gone quite differently. It may still have failed, but it would have failed much faster and according to an actual strategy—which could then be refactored for lessons learned.

When an organization stands poised at the first step of a migration journey, it is important that the leaders, well, lead the way. Because when the cloud native destination is reached, the company will not be the same company. It will build in a different way, compete in a different way, generally behave in a completely new way. And because change is scary, the CEO and board need to lead the way with confidence.

That is why the moment where Executive Commitment is achieved is the moment a transformation truly begins.

Vision and Core Team

Having named a Transformation Champion (fortunately, WealthGrid was smart enough to officially name Jenny to the role she had been filling all along), established the compelling need for cloud native with Business Case, and achieved Executive Commitment for making it actually happen, it's now time to plan the actual transition.

Two things need to happen now: we need to create a proper migration strategy and pick the team to deliver it. In its first attempt to go cloud native, WealthGrid had neither. Not on purpose, but because it simply made the same mistake most companies do when launching a transformation: simply adding the transformation to the usual Agile backlog of tasks. Functionally, this means tasking an existing team with building the new system while still being responsible for their regular duties, too. Jenny's team never had the chance to build much of anything at all. But even when a team does get more time to work on it than they did, or has simply a long timeframe to do it in, they still won't be able to deliver. Limited experience coupled with a lack of space and flexibility for research leads to pursuing cloud native implementation using "well-known ways"—that is, trying to deliver cloud native using incompatible approaches from Waterfall, Agile, or a combination of the two. As we have seen, the initiative falters.

Two cognitive bias factors influencing things here are the ambiguity effect, where people facing a situation with a lot of unknowns will fall back on simply doing what they already know how to do, using methods that have worked in the past. Similarly, law of the instrument leads them to choose tools that they know, rather than seeking the right ones for the job. When there is time pressure and little or no opportunity to explore new tools and techniques, guess what is going to happen? Yes, people will pick up their comfortable tools without even thinking.

Furthermore, without an overall consistent vision and a core team in charge of delivering it, different teams will make independent and frequently conflicting architectural decisions. This is exactly what we saw happen with WealthGrid's second attempt.

To avoid both of these problems, we use the Vision First, Objective Setting, and Core Team patterns to set the correct path right away. These mark a gradual shift away from abstract strategy toward concrete execution, which moves down the chain to middle management and the tech teams. But first, before we leave Steve and the rest of WealthGrid's C-suite execs, we have one more pattern for them: Decide Closest to the Action.

Delegating Power

Traditional organizations are all about delegating responsibility, but not authority. Important execution decisions are rarely if ever delegated to those actually in charge of carrying them out. This pattern is about putting the decision close to where the change is happening, and doing this is key to successfully evolving to a cloud native culture.

In the past, the flow went: strategy -> vision -> execution. All planning happened at the first step and almost always at the top of the chain of hierarchy. Once you moved on to vision, planning was done; strategy was set and execution was to follow. Static strategy doesn't work in cloud native, as we have seen with the Dynamic Strategy pattern.

In a way, the Decide Closest to the Action pattern is about dynamic planning and delegation. The simple (but not easy) idea behind it is for decisions to be made at the point of action by the person who's going to be doing the action. So, if your team is responsible for delivering a particular microservice, it is your responsibility to do the planning around that microservice. Not your boss, not the project manager, not the systems architect, certainly not the folks several flights up in the nice offices. *You*.

This is where this pattern perhaps gets a bit tricky to implement. In traditional organizations, managers typically do the vast majority of planning and decision making. Then they hand down a set of pre-specified tasks to the engineering teams to execute. Cloud native velocity, however, depends on teams being able to make fast decisions regarding whatever task they are working on, without needing to consult the boss. The managers have to be willing to give up their traditional position of control in order for this to happen.

This is important so we will say it again: in cloud native, the engineers get to plan and to make decisions regarding what they're working on. They don't get handed a set of marching orders from a project manager to fulfill exactly as specified. They don't

need to seek permission to try something different if they spot a better way to go. In cloud native, the power to decide rests with the person doing the job.

PATTERN: DECIDE CLOSEST TO THE ACTION

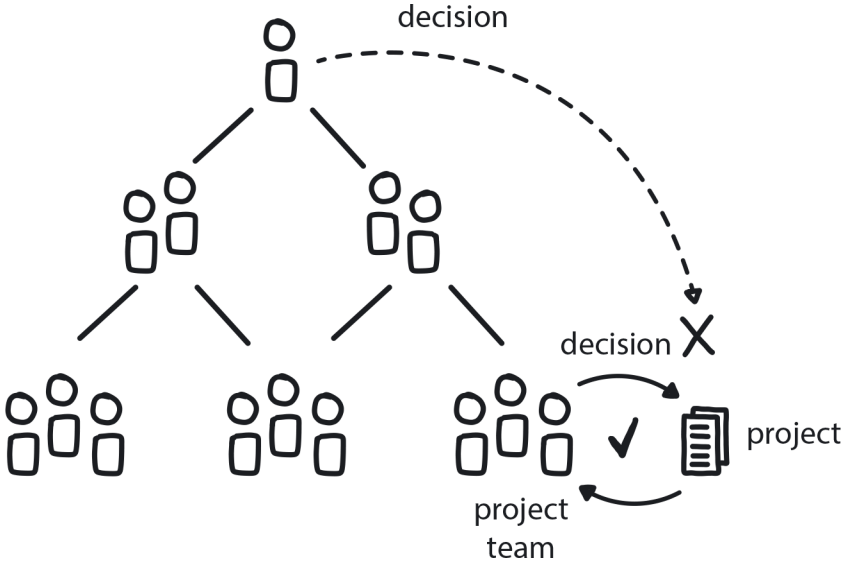


Figure 2-5. Decide Closest to the Action

Those closest to a change action get the first chance to make any decisions related to it.

Like Dynamic Strategy, Decide Closest to the Action is a super pattern that applies not just in every phase of the transformation but permanently across the entire organization. Paired closely with Decide Closest to the Action is another super pattern, Psychological Safety. This addresses the need for people throughout the organization to feel they can express ideas, concerns, or mistakes without being punished or humiliated for speaking up.

In a work environment that lacks Psychological Safety, team members hesitate to offer new or different ideas, especially when these may diverge from the overall group opinion, and will generally act in risk-averse ways. Since cloud native depends on teams collaborating to come up with innovative solutions through independent, divergent thinking, Psychological Safety is essential for creating an environment that makes this kind of co-creative work possible.

If teams lack the ability to fail safely when experimenting, they waste time exhaustively analyzing all future outcomes to minimize exposing themselves to risk, effectively killing both creativity and velocity. In a psychologically safe work environment, though, people are willing and even excited to try, possibly fail, and then try again as they strive to come up with the next idea (which might be wild, but also possibly game-changing). And they will be able to make decisions with confidence, knowing that their team will tell them—honestly yet constructively—when they might be making the wrong one.

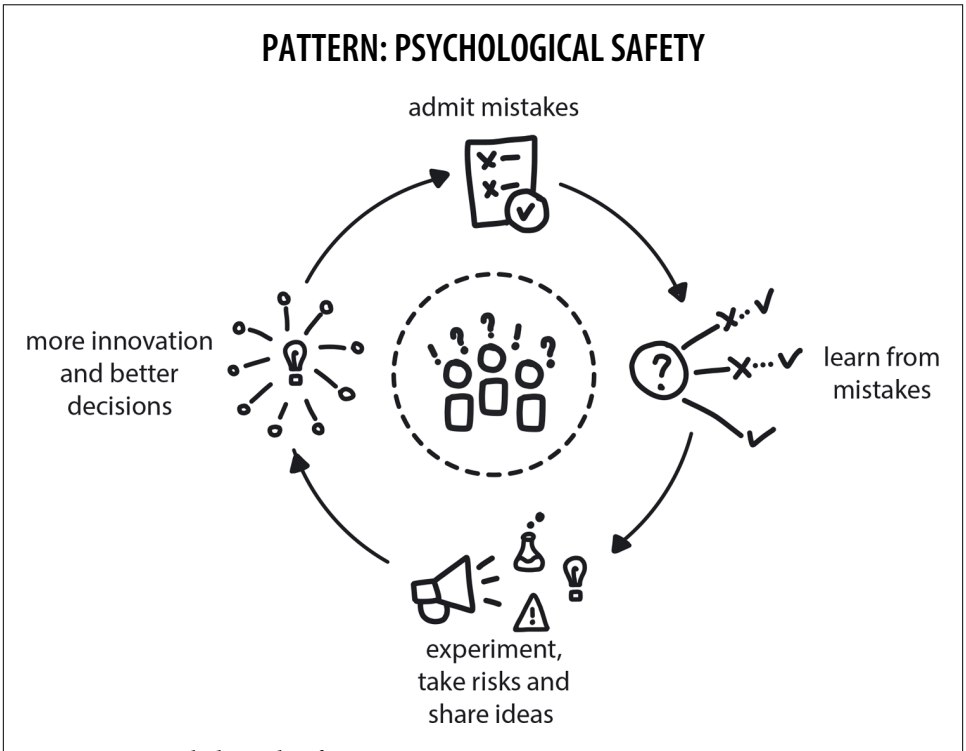


Figure 2-6. Psychological Safety

When team members feel they can speak up, express concern, and make mistakes without facing punishment or ridicule, they are able to think freely and creatively, and to be open to taking risks.

Once these innovation-empowering cultural patterns are put in place, the company is now ready to take its first concrete step toward the cloud native transformation: naming the Core Team.

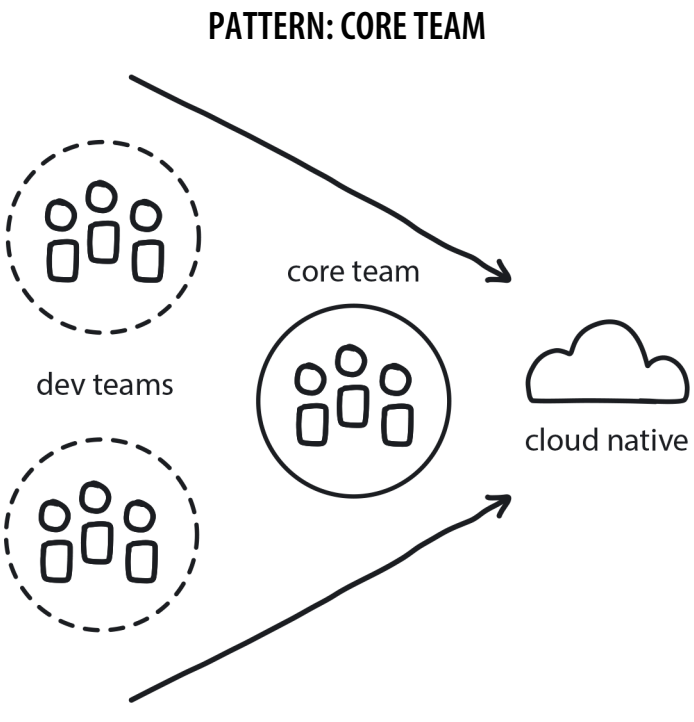


Figure 2-7. Core Team

A dedicated team of engineers and architects, constantly diving deeper into technical challenges, will reduce the risk of the transformation. The team members gain experience that will help onboard the remaining teams more quickly and effectively.

Creating the Core Team means handpicking a relatively small team to drive the overall innovation and, especially, take charge of the initial stages.

What does a Core Team look like? Well, it's compact—typically, five to eight engineers (including software architects, if you've got them to draw from). Team responsibilities will include ownership of the technical vision and architecture, de-risking the transformation by running a series of Proofs of Concepts (PoCs), creation of a Minimum Viable Product (MVP) version of the platform, and later onboarding and guiding other teams. (No worries, we've got patterns coming for all these.)

Essentially, the Core Team rapidly iterates through the most challenging parts of the transformation. They begin by establishing a high-level vision and then translating the vision into concrete, executable objectives. They use research and experimentation to build their own knowledge and experience in the cloud native paradigm—using the knowledge they gain to adjust the transformation vision and architecture as they go. Later, the Core Team's first-hand understanding helps them to onboard other teams to the new way of working. Their knowledge paves the path for the rest of the company for a seamless and successful cloud native adoption.

The team often stays together even once the transformation is done, because they have effectively become the main repository for the organization's cloud native knowledge. Sometimes the Core Team, or perhaps a portion of the team, forms the Platform Team (more on that soon). Or, once the new system is running fully in production mode, they may remain in charge of continually improving the platform—remaining in an innovation-and-research mode to help the company stay ready for whatever future comes next.

Once chosen, the Core Team's very first task is Vision First, outlining a high-level transformation plan.

PATTERN: VISION FIRST

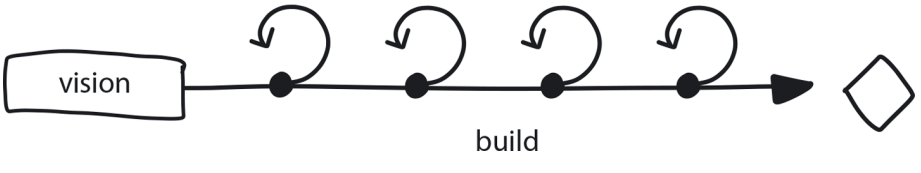


Figure 2-8. Vision First

Defining a guiding vision as very first step helps set the right path through an uncertain environment.

In Vision First, the Core Team needs to define a clear and achievable vision that can be translated into specific executable steps. This vision defines and visualizes the architecture of the whole system up front. Since the Core Team is likely still themselves early in the cloud native learning curve, this vision can be created with the help of external resources, such as consultants. This will definitely help the vision stage move quickly.

If time allows, the Core Team can work to find their own way, uncovering the vision/design through a series of small research and prototyping projects, ranging from a few hours to a few days each. This DIY approach is ideal, since it hastens the learning that the Core Team needs to undergo anyway, but it can make this initial phase of the

transformation take longer to complete. An experienced consultant can help jump-start that learning, though, while steering things in the right direction, so perhaps a combination of the two is the best of both worlds.

No matter which way you get there, Vision First outlines the technical and organizational roadmap for the company for the next year or two. As the team members work to create this vision, they need to remember that this is not carved in stone—vision needs to be dynamic, just like strategy. At this point, they don't have the full picture of what they are trying to achieve and the initial vision is just their best guess at the very beginning. Don't get stuck on this early idea, because it needs to evolve as the transformation progresses and new information is uncovered.

The question to answer is, essentially, *Where are we going?* It's important to keep the answers high level enough to allow freedom of choice during implementation (in other words, don't start choosing tools just yet). At the same time, though, things need to be detailed enough to provide clear guidance, which will help avoid common pitfalls.

The practical implementation of creating a cloud native transformation vision is, *do* name names—of architectural elements. *Don't* start choosing specific tools or service providers at this stage.

Good things to include
: Semi-technical terms describing broad concepts, such as "microservices" and "containerized platform in the cloud."

Things to avoid:
: Committing to any specific tool or tech, no matter what kind of great things you've heard about it. (Don't worry, we will be getting to that very soon, and by way of applying patterns to help make sure your choices are the right ones.)

Once the Core Team lays out the Vision First pattern, it moves quickly and directly to translate it into concrete objectives. This is where we begin to create specific architecture and a series of steps to move toward the Maturity Matrix cloud native standard.

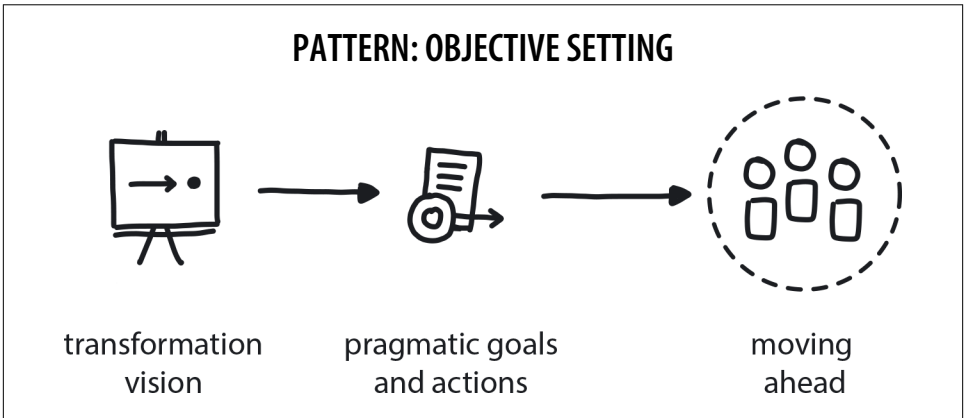


Figure 2-9. Objective Setting

After establishing a transformation vision, the next step is to translate it into pragmatic goals and actions for moving the initiative ahead.

The Objective Setting pattern marks a pivotal point, because this is where we move from theoretical strategic planning to defining concrete, pragmatic, and above all executable steps toward the cloud native goal line.

The whole thing began with a strategic, executive decision-making phase, which—aside from the Transformation Champion, who can come from any part of the organization—was chiefly in the hands of the company's senior leaders, CEO, and board. The journey there was abstract, moving through establishing a case for and then committing to a transformation initiative. The key outcome was to make a firm decision: *We want to be cloud native and we are willing to provide the resources to make it happen.*

The Core Team was then created to begin turning strategy into reality. First, defining the vision—the high-level roadmap for the transformation—and then translating that into an architecture and a set of achievable objectives: *We want DevOps teams structure, need to refactor into containers and microservices, and want to install an orchestration platform to manage everything.*

The time has come for creating the steps that will pragmatically achieve these objectives: A framework of tasks to achieve specific goals—first we need a basic platform, so we will do experiments, try some PoC—that becomes execution. Figure 2-10 shows the patterns that we've followed thus far.

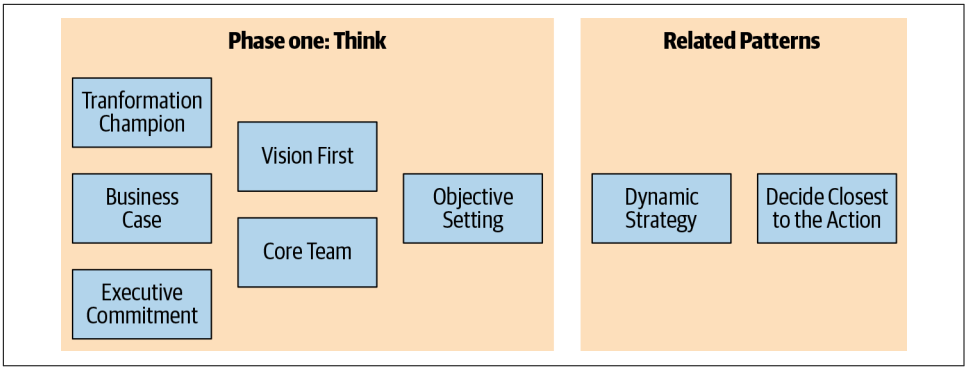


Figure 2-10. Phase 1: Think, the first step in our cloud native transformation design, covers strategy and objective setting.

Thus have we progressed from idea to strategy to objectives, which brings us to the end of Phase 1. It's time to move into Phase 2: Design, where we begin to identify the elements for constructing our cloud native platform.

PHASE 2: DESIGN

Phase 1 gave us a first, rudimentary roadmap for finding our way to the cloud: a high-level vision of Microservices Architecture delivering Containerized Apps in the cloud by Build-Run Teams. Wait, what are all those things?! Don't panic: these are all patterns that, along with others, will carry us from vision to delivery. Let's take a look.

The pivot point between Phase 1 and Phase 2 consists of splitting the organization, at least temporarily. The Core Team is setting off to design and build the new platform, while the rest of the company carries on with business as usual for the time being.

These two paths require separate sets of patterns and separate approaches. The Core Team is managed as a creative team, given a purpose and the freedom to experiment and research their way to answers. The rest of the teams still keep building the value in the existing system, until the new platform is ready. They are managed for proficiency.

Not to give short shrift to the proficiency teams—they are busy making money for the company while the Core Team innovates—but they only get one pattern for the time being. Most teams will eventually need to evolve to cloud native's more flexible, intuitive, and problem-solving way of working. During the transition, however, it's important to keep the proficient teams proficient and run them as they have always been run, not changing to cloud native techniques yet. Since they aren't changing

much (yet—their time is coming soon!) the only thing to discuss right now is how best to manage them in the interim.

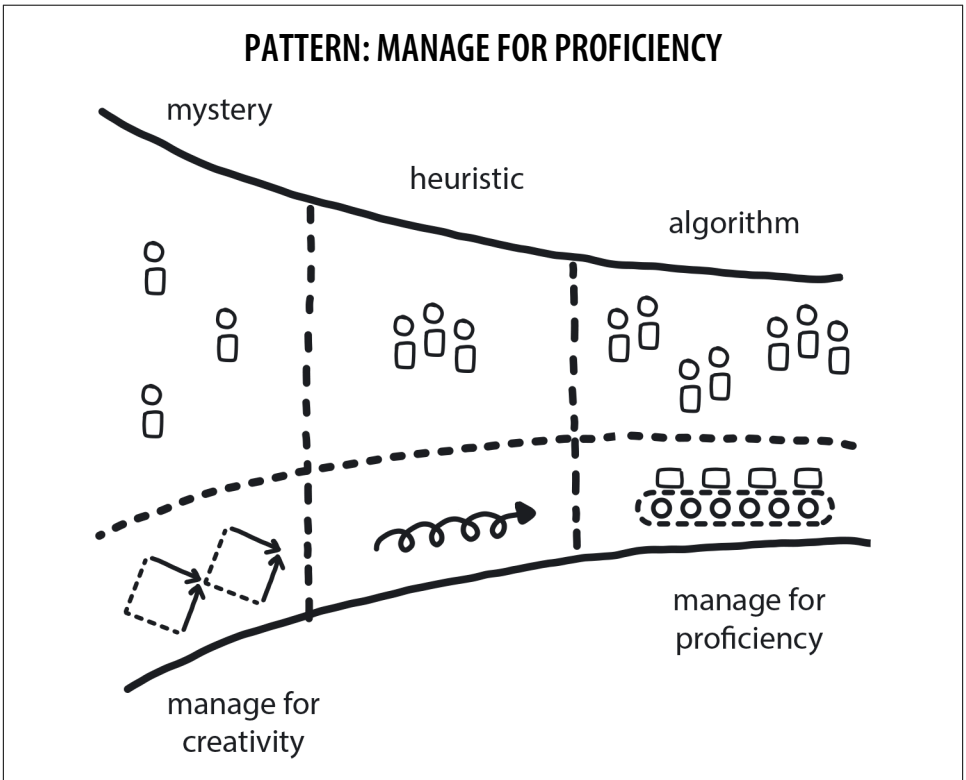


Figure 2-11. Manage for Proficiency

Teams delivering stable and highly repetitive or algorithmic work should be managed for high quality and optimal efficiency.

We have named this the Design phase, but it could just as easily be called the Exploration or Innovation phase. Now is the time that the Core Team delves into creative research, and they need to be appropriately managed for producing innovation.

PATTERN: MANAGE FOR CREATIVITY

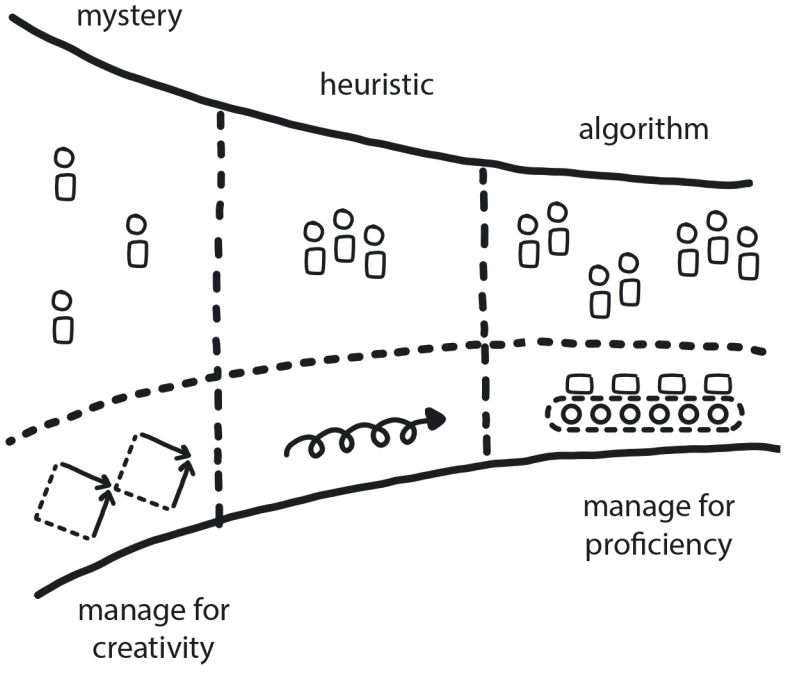


Figure 2-12. Manage for Creativity

Teams charged with innovation need the open-ended freedom to experiment their way to solutions without pressure for delivering specific results on a set schedule—and the freedom to sometimes fail along the way.

In their first attempt to go cloud native, WealthGrid made the extremely common mistake of having an existing team work on building the new system while still responsible for their regular duties, too. The obvious problem is that urgent issues on the existing system—the one that is making money for the company—will always take priority over building something new that isn't going to add value for who knows how long. For that reason it makes sense that the Core Team should do nothing but focus on the transformation. It also makes sense that this team is going to need some intense time investment for learning and experimentation, so they shouldn't be distracted by unrelated work.

The most important reason that the Core Team needs to stand alone, however, is because they are breaking away from a longstanding way of doing things in order to do something completely new and different. This means that the team needs to work, and be managed in their work, in a completely new and different way.

Remember all that talk about proficiency vs. creativity? The Core Team needs open-ended time to work, and this can be a challenge for others in the organization. Managers are used to overseeing responsibilities you can, well, manage, in a predictable, top-down, hands-on way. Managing for creativity is more about facilitating innovation: providing the opportunity for innovation to incubate by defining the goal, providing the proper circumstances, and then stepping back. You cannot manage facilitation.

Practically speaking, how does the Core Team start to make smart design choices for this new cloud native system while minimizing risk? Gradually Raising the Stakes.

PATTERN: GRADUALLY RAISING THE STAKES

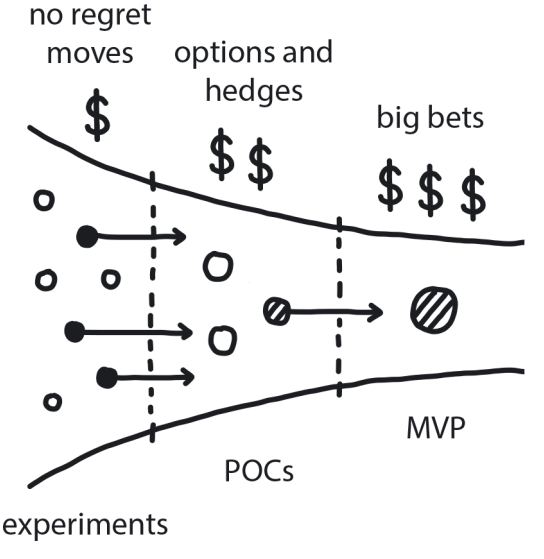


Figure 2-13. Gradually Raising the Stakes

In an uncertain environment, slowly increase investment into learning and information-gathering actions; eventually you uncover enough information to reduce risk and make better-informed decisions.

Related sub-patterns: No Regret Moves, Options and Hedges, Big Bet

In the predictive world of Waterfall, when you launch a big project, you can typically make big decisions right away. The project itself might be new, but it's almost certainly going to be a lot like previous initiatives. Now, however, the Core Team is feeling their way onto the cloud with only a very basic map. The next move is far from clear because they are still early in the process of building their own cloud native knowledge, and low-certainty projects carry extremely high risk. Any decision made early on in this uncertain environment is highly likely to turn out to have been a wrong choice.

Gradually Raising the Stakes is a super pattern and a general strategy for cloud native risk management that is critically applicable to this pivotal point in a transformation. This pattern gives us the tools to address uncertainty by moving step by logical next step. The way it works is through three graduated levels of risk taking: little risk, moderate risk, high risk. In terms of pure strategy formation these are the subpatterns No Regret Moves, Options & Hedges, and Big Bet.

When it comes to creating a transformation design, however, these patterns correlate with three technical patterns. Applied in proper order, these are practical steps for reducing transformation risk: Exploratory Experiments, PoCs (Proof of Concepts), and Platform MVP (Minimum Viable Product). We will encounter these patterns now, along with other patterns that are important at this stage of the initiative. These are mainly around adopting new, cloud native-oriented ways of thinking and doing things.

The gateway to these is the Distributed Systems super pattern, because the rest of the transformation is essentially implementing an optimized cloud native version of distributed systems architecture. The Exploratory Experiments pattern, to establish the initial framework for our cloud native explorations. We then unpack Distributed Systems super pattern and all the related pattern puzzle pieces the Core Team will get to play with.

Distributed Systems and Friends

Welcome back to the most technical subject matter covered in this book: the broad technologies and methodologies that together compose cloud native architecture. The Core Team is experimenting to find the best implementation of each pattern for their particular transformation initiative—a level of technical detail we are saving for the

sequel to this book—but the pillars are unchanging. It all begins with Distributed Systems.

PATTERN: DISTRIBUTED SYSTEMS

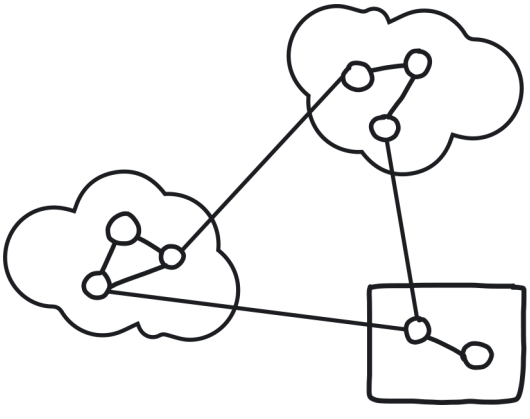


Figure 2-14. Distributed Systems

When software is built as a series of fully independent, loosely coupled services, the resulting system is, by design, very fast, resilient, and highly scalable.

The development of a monolithic software system is limited by its sheer complexity, which only a few top-level architects and managers may be able to comprehend (but even this is not guaranteed). Indeed, it is tempting to start a new software project as a monolith to get things started quickly, since the architecture is simple, and the few components share resources in a single codebase.

As the system expands, however, the size and complexity of its major components keep growing. Over time, this causes a variety of bottlenecks to develop, mainly in the form of dependencies on all different levels: data sharing, calling functions from other components, and teams being held up by other teams. All this leads to an increasing level of coordination required to produce any change, to the point where the overhead of coordination becomes larger than the work itself. We have seen systems that require three days and seven different teams to release a single small change that took three hours to create.

All of this eventually makes further system growth functionally impossible. The main, maybe only, fix would be to scale the whole system, even though only a small part of it is running out of resources—which requires buying very expensive hardware. (And then, over time, the same problems would simply re-emerge anyway.)

Distributed systems are much more complex to initially architect and implement, but once that initial work is invested they are much simpler to grow and evolve forward with improvements, new features, and changes. This is due to a much lower level of coordination overhead. That same three-hour code change task can simply proceed to production in 10 minutes without any human involvement at all.

A distributed system is any system whose components are executing independently, communicating and coordinating their actions by passing messages to one another. Historically these were on-prem servers, but systems began moving to the cloud. Cloud native then emerged as a software development approach that makes full and optimal use of the cloud computing model. Cloud native applications are built with modular service components—microservices—that communicate through APIs and get packaged in containers. The containers themselves are managed via an orchestrator (Kubernetes) and dynamically deployed on an elastic cloud infrastructure through Agile DevOps processes and continuous integration/continuous delivery (CI/CD) workflows.

OK, so that was a lot. Don't worry, it's only the general concepts we need to worry about right now—the high-level master blueprint for the system under construction. This leads us, aptly enough, to literally drawing one up.

PATTERN: ARCHITECTURE DRAWING

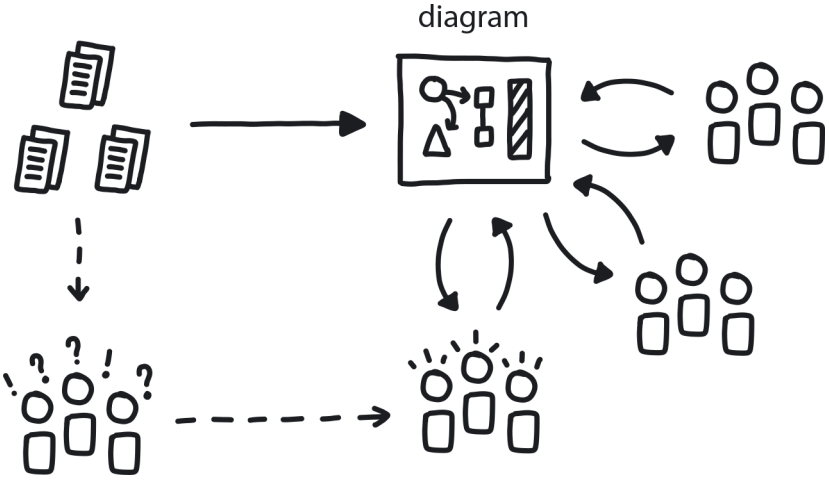


Figure 2-15. Architecture Drawing

A picture—or, in this case, a high-level outline sketch of your system's basic architecture—can replace a thousand words, save time, and prevent misunderstandings.

Distributed systems are inherently complex, which makes them difficult to visualize and describe, much less discuss. The Architecture Drawing is a classic No Regret Move, easy and inexpensive yet extremely beneficial. It's also, thankfully, straightforward to do: produce a single, unified, and official version of the new cloud native system's architecture for circulation and reference. This creates a simple and consistent visual that everyone on the Core Team should be able to draw, from memory, in less than 30 seconds.

A huge benefit to the Architecture Drawing is that it helps all stakeholders internalize a deep understanding of the architecture's components and how they relate to each other. It also provides consistency of thought and makes discussion vastly easier.

With this clear blueprint of our distributed cloud native system in hand, let's look at the patterns that comprise it. Figure 2-16 shows the technical, structural, and process-oriented patterns that go into designing cloud native distributed systems and underlie the Distributed Systems pattern:

- Microservices Architecture and Containerized Apps are fundamental pillars of cloud native architecture.
- Dynamic Scheduling, Automated Infrastructure, Automated Testing, and Observability describe these core technical aspects of a cloud native system.
- Continuous Integration (CI) and Continuous Delivery (CD) are the processes for implementing them.
- Secure From the Start Systems addresses how to keep everything safe.
- Reproducible Development Environment describes the workspace necessary for the Core Team to assemble all these pieces.

Thumbnail versions of all these patterns are available in the Appendix: Library of Patterns, which will direct readers to the full versions of each pattern in this book. Since we have discussed all of these technical aspects, particularly microservices, containers, and dynamic scheduling (a.k.a. Kubernetes), throughout the first part of this book, let's leave the nitty-gritty details to the Core Team and keep the transformation design moving along at the level of architecture rather than detailed execution.

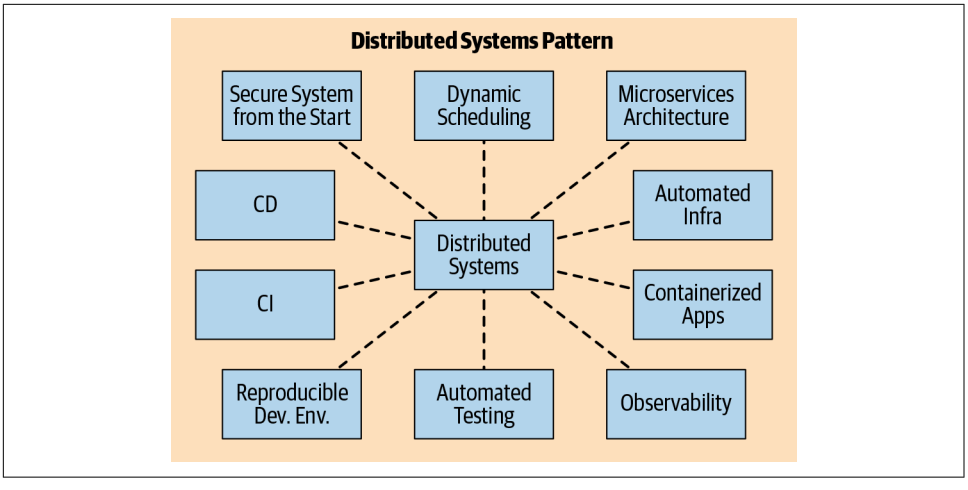


Figure 2-16. The Distributed Systems pattern unpacks into essential elements of the cloud native architecture and development process.

Exploratory Experiments

This is the practical first step in an organization's technical transformation. At this point, the Core Team doesn't yet have enough cloud native knowledge to identify the most general direction to head in—you can't research unknown unknowns. So they begin by doing theoretical research: reading what others have done, case studies, white papers. Certainly doing No Regret Moves like trainings and other learning opportunities. There is no clear solution available because this is a complex problem and you cannot simply Google it—there are many solutions out there that sound great but simply don't fit your particular use case.

The scope of the transformation is very wide. You need to collect many different puzzle pieces to create a consistent platform, and right now you don't even know enough to understand what the right pieces even look like.

How do you find the right pieces? By doing experiments. Small, quick, low-cost investigations into different tools, techniques, and architectural combinations just to see what works and, more to the point, what does not. Core Team members learn enormously as they do this, building their understanding and skills. After a relatively short time, usually a month or two, this new knowledge and the experimentation pro-

cess helps the team narrow down the field of options and uncover likely best-fit directions for meeting the transformation objectives.

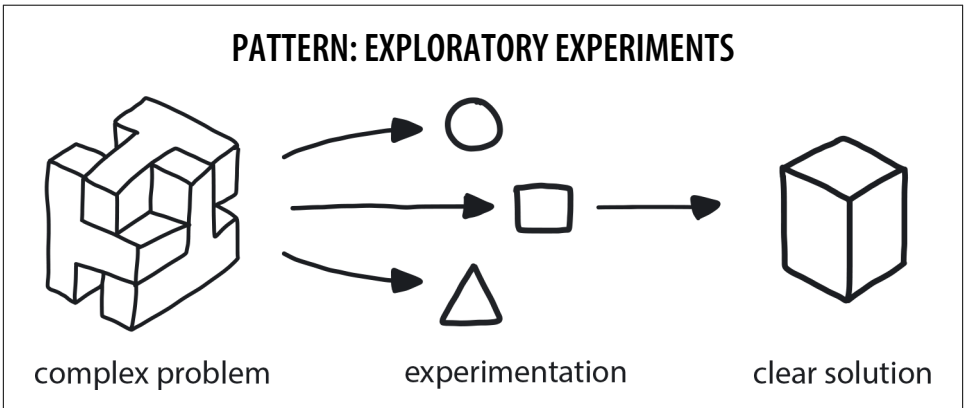


Figure 2-17. Exploratory Experiments

When dealing with a complex problem with no obvious available solution, run a series of small experiments to evaluate the possible alternatives and learn by doing.

This is the time of working with the fundamental building blocks of cloud native: Microservices Architecture, Containerized Apps, Continuous Integration, Continuous Delivery, and different types of automation. We will start with Microservices Architecture, because that drives everything else.

PATTERN: MICROSERVICES ARCHITECTURE

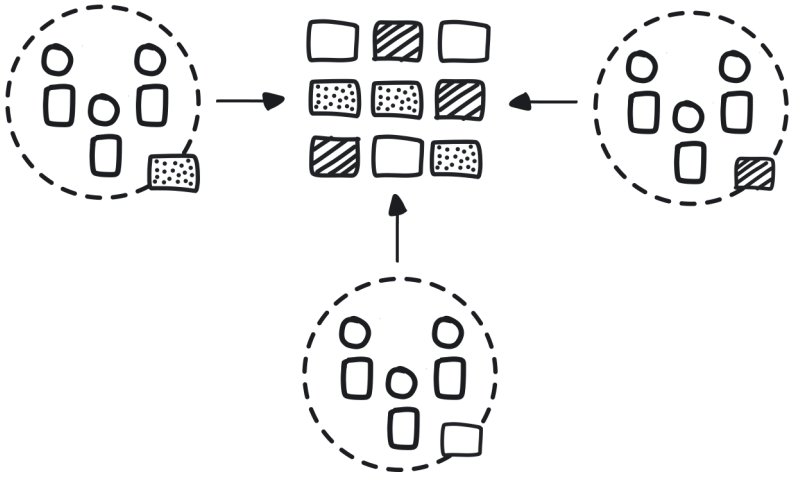


Figure 2-18. Microservices Architecture

To reduce the costs of coordination between teams delivering large monolithic applications, design software as a suite of modular services that are built, deployed, and operated independently.

Microservices are an implementation of distributed-systems architecture where an application is delivered as a set of small, modular components that are completely independent from each other and communicate via APIs. This is an extremely flexible, resilient, and scalable way to build software, but also a very complicated one. In the cloud native way of working, a team is responsible for delivering one service, from design to delivery, and then supporting it as it runs even as they iterate forward with small improvements.

To run the application, all of its microservices get packaged into a container. Containers are lightweight, standalone executable software packages that include everything required to run an application: code, runtime, system tools, libraries, and settings.

They are a sort of "standard unit" of software that packages up the code with all its dependencies so it can run anywhere, in any computing environment.

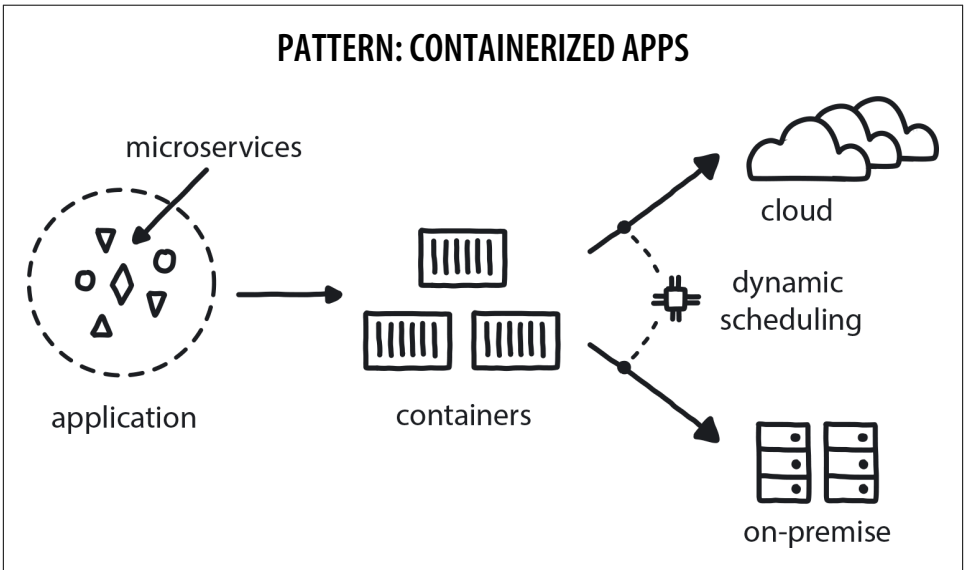


Figure 2-19. Containerized Apps

When an application is packaged with all its necessary dependencies, it does not rely on the runtime environment and so can run agnostically on any platform.

To run in the cloud, whether public or private, the containerized microservices need to be organized, managed, and told when and where to run. This is the job of the Dynamic Scheduler, or orchestrator (Kubernetes is by far the most well known and widely used but is far from the only orchestration option out there).

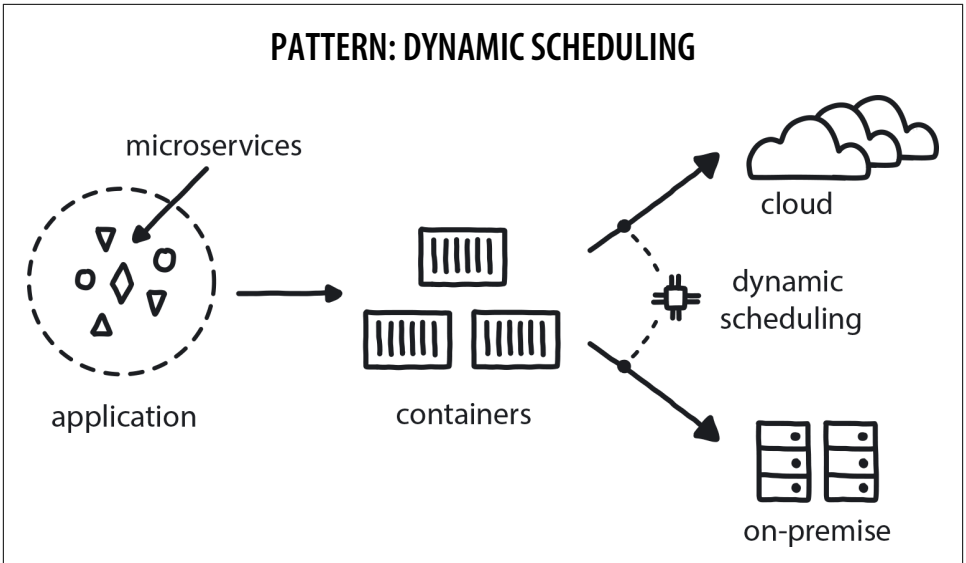


Figure 2-20. Dynamic Scheduling

An orchestrator (typically Kubernetes) is needed to organize the deployment and management of microservices in a distributed, container-based application to assign them across random machines at the instant of execution.

Building software that is optimized for the cloud is complicated, but it does come with a few helpful advantages. Since there are so many moving pieces now, far more than humans can manage or keep up with, cloud native computing requires full automation. Automated Infrastructure handles server provisioning, configuration management, builds, and deployments and monitoring at top speed and without the need for manual intervention. Similarly, Automated Testing ensures that all the testing required to take any product change to production is built into the development process.

PATTERN: AUTOMATED INFRASTRUCTURE

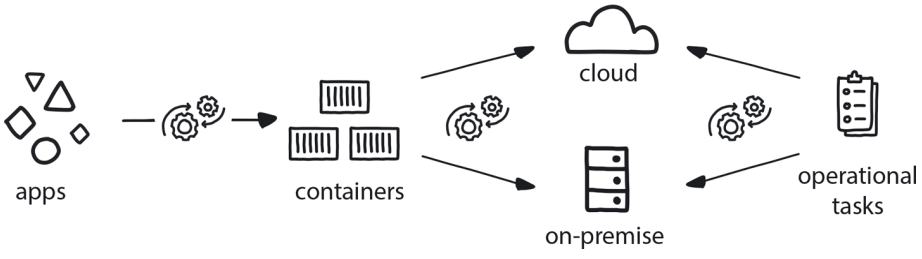


Figure 2-21. Automated Infrastructure

The absolute majority of operational tasks need to be automated. Automation reduces inter-team dependencies, which allows faster experimentation and leads in turn to higher development velocity.aut

PATTERN: AUTOMATED TESTING

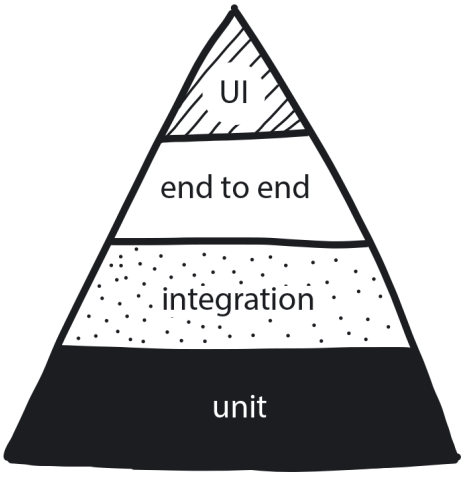


Figure 2-22. Automated Testing

Shift responsibility for testing from humans (manual) to an automated testing framework so developers can deliver faster and have more time to focus on improving features to meet customer needs.

To make the most of automation, the Core Team must adopt Continuous Integration and Continuous Delivery as their workflow.

Continuous Integration (CI) is a coding philosophy and set of practices that drive development teams to implement small changes and check in code to version-control repositories frequently. The technical goal of CI is to establish a consistent and automated way to build, package, and test applications that are always highest quality, and always ready to be released.

PATTERN: CONTINUOUS INTEGRATION

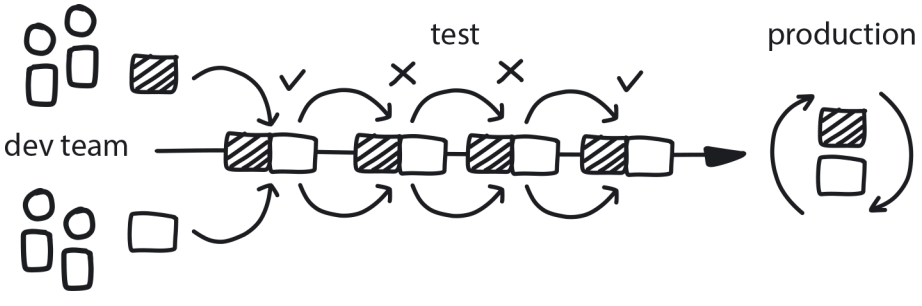


Figure 2-23. Contiguous Integration

Frequent integration of small iterative changes speeds overall delivery and increases the quality of the code.

Continuous Delivery(CD) starts where Continuous Integration ends. CD automates the delivery of applications to selected infrastructure environments. Most teams work with multiple environments outside the production pipeline, such as development and testing environments, and CD ensures there is an automated way to push code

changes to them. CD automation then performs any necessary service calls to web servers and databases, and executes procedures when applications are deployed.

PATTERN: CONTINUOUS DELIVERY

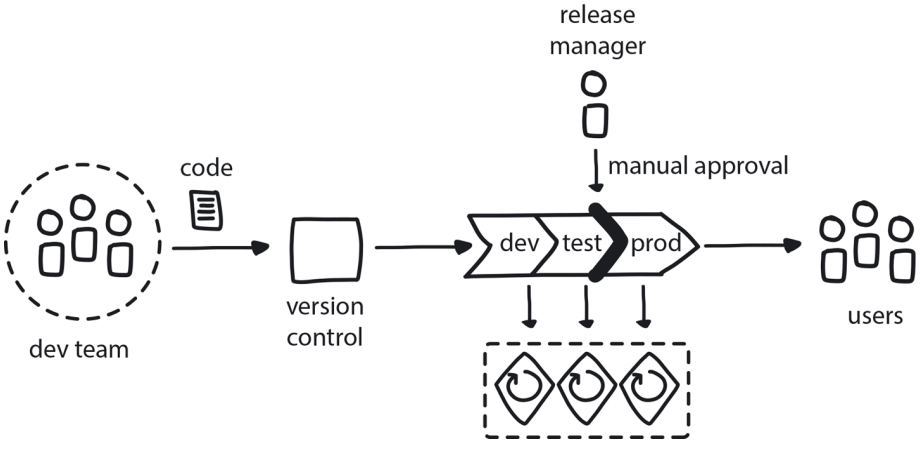


Figure 2-24. Continuous Delivery

Keeping a short build/test cycle means code is always ready for production and features can be immediately released to customers—and their feedback quickly returned to developers.

Better together: Continuous Integration and Continuous Delivery are so integral both to each other and to the cloud native approach that they are usually used as a single term. CI/CD used together make it possible to implement continuous deployment, where application changes run through the CI/CD pipeline. Builds that pass Automated Testing get deployed directly to production environments with Automated Infrastructure. This happens frequently, on a daily—or even hourly—schedule.

CI and CD are difficult to achieve, however, unless developers are able to test each change thoroughly. The Reproducible Dev Environment pattern defines the condi-

tions for where the Core Team can practice all of these brand new tools and techniques while applying CI/CD processes.

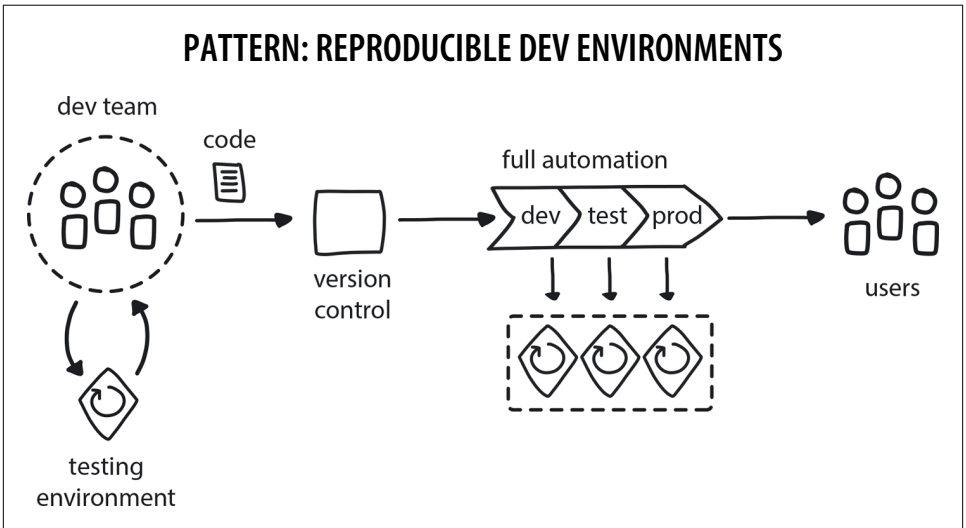


Figure 2-25. Reproducible Dev Environments

Developers need to test their daily work in an environment that is easy to spin up and that matches production tooling as closely as possible.

The Core Team has now learned how to orchestrate the containerized microservices they've been experimenting with, using automation and cloud native methodologies. What comes next?

Proof of Concept

So, great! Your research has given you a better understanding of what is going on, but there is still fairly high risk involved in making decisions at this point. How do you know when it's time to stop Exploratory Experiments and take the next step?

The trick is to look at it as a balance situation. You are weighing the need for more certainty against the need to move things forward. The moment of truth comes when a few likely looking paths begin to stand out from the cloud native wilderness of choices. You still need to gather more information—a lot more—but you have achieved the milestone of identifying some promising areas where you need your information to be deeper.

At this point it's time to move to the next stage of the process: deepening understanding of the most promising areas. It's time for the Proof of Concept (PoC) pattern.

PATTERN: PROOF OF CONCEPT (PoC)

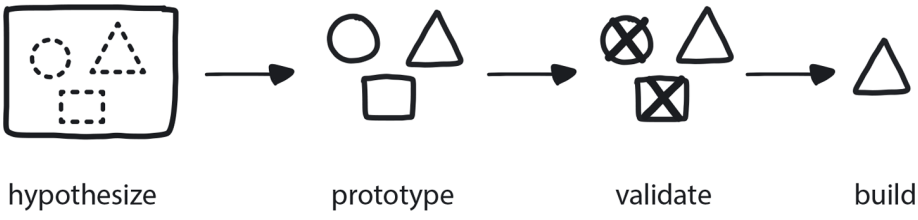


Figure 2-26. Proof of Concept

Before fully committing to a solution that can significantly affect the future, build a small prototype to demonstrate viability and gain understanding.

You still don't know enough to make any kind of large commitment at this point, whether to a vendor or an open-source solution. Any decision right now carries massive risk because full commitment to one path means you will continue to build further functionality on top of this solution. The further you go, the harder it becomes to change course.

You have, however, narrowed the field to a few potential transformation paths that appear promising. It is time to investigate these different options for architecture and toolsets through deeper, though still not prohibitively difficult, Proof of Concept exercises. The stakes are a little higher now: PoCs are more complicated than the simple experiments the Core Team has been running thus far, so costs and time investment do go up. Thus, if your best-guess PoC fails there is some pain, but it's not a terrible tragedy. You can revert if needed (not to mention gain valuable insight into what doesn't work).

PoCs are fun! A real chance for the engineers to roll up their sleeves and play with these cool cloud native toys by choosing one of the potential solutions the Core Team has identified through experiments and building a basic prototype to demonstrate viability—or lack thereof. Remember that it's still OK to fail.

PoCs: Eliminating Weak Options

Something to keep in mind during the PoCs phase is how young the cloud native ecosystem still is. Even now there is still a lot of uncertainty and knowledge to be gained, and there are also still significant gaps in the toolset. Cloud native right now is similar to the birth of the automotive industry at the beginning of the 20th century, when there were more than 100 automobile manufacturers in the United States alone. Now there are fewer than 20 major manufacturers worldwide, and who has heard of most of the ones that dropped away?

So, there will be shakeout and consolidation. Every month major new tools get introduced into the cloud native ecosystem, and they don't always fit each other well. The entire point of PoCs is to choose 10 to 20 of those tools and try to see how they work together. Among the safest bets to choose from are in the Cloud Native Computing Foundation's *"Graduated Projects"*. These are well-established tools like Kubernetes, which are likely to be around for the long run, and there are many related/supporting tools and resources built around them.

Quick and dirty is the theme with PoCs, which should take a few days to a few weeks at most to build the most primitive version you can get away with. Focus only on hard issues related to the specific needs of the current project, and stop when the solution is clear—don't worry about creating a nice UX, just make the darn thing work. If possible, try to actually demonstrate something functional so you can collect information about how this solution will affect the business.

But, most of all, be ready to throw it away when you are finished proving your point. This doesn't mean that you must absolutely trash all your hard work. But you probably should. Very often when initial quality is intentionally low, it's cheaper and easier to just scrap it and rebuild. This way you get a clean slate and the chance to do things right, applying all those hard-earned PoC lessons you just learned. Otherwise, you will probably have to spend a lot of time fixing all the stuff you did wrong before you can even start building anything new on top.

Figure 2-27 shows the map of patterns we've covered in Phase 2: Design, from the point where the Core Team began researching the best possible design for the company's cloud native transformation. The team members are being managed for creativity, with freedom to experiment and fail if necessary, while the rest of the company keeps on delivering on the existing system. The Core Team does Exploratory Experiments to investigate all the aspects of Distributed Systems that will go into the new cloud native platform, until their learning curve has flattened and they've identified some good candidates for more in-depth investigation through PoCs.

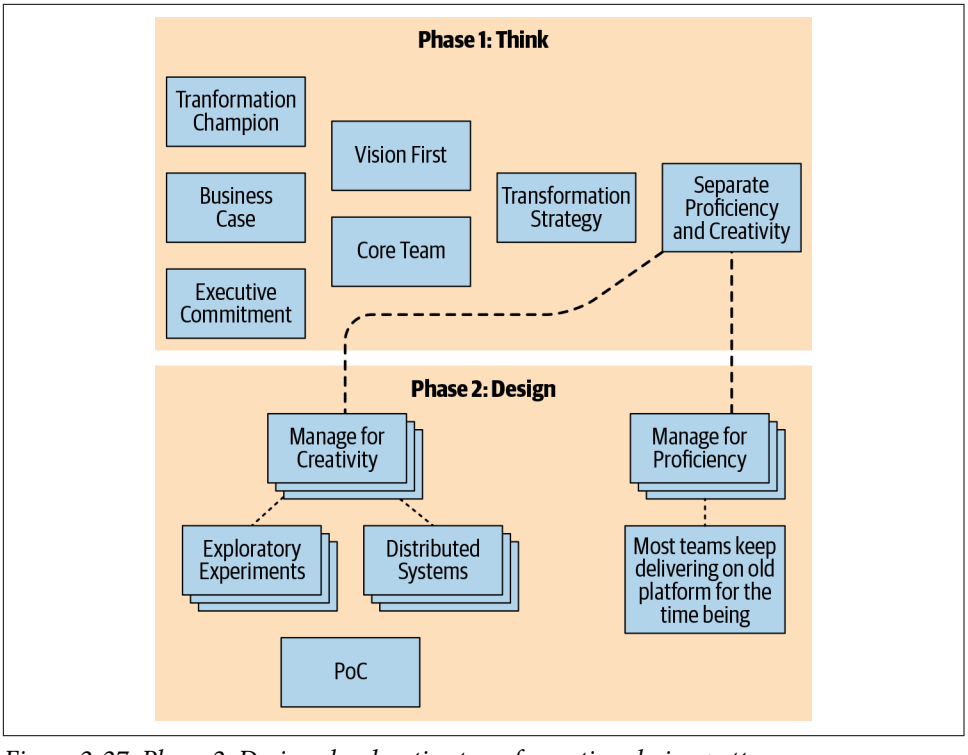


Figure 2-27. Phase 2: Design cloud native transformation design patterns

Because once you've picked a clear winner—once the PoC process has mapped the likely right transformation path—it's time to get ready to build for real!

So far in our design for a cloud native transformation, we have covered getting ready for the transition (including figuring out if you really even need to do one in the first place!). Next we had a detailed look at how to de-risk the transformation by experimenting your way to the right path, using a gradual and iterative approach. Now that PoCs have revealed the likely best platform, we will move on to Part 2 for applying patterns to do the actual implementation.

Epilogue

We have reached a point that would usually count as the finish line for a story: the happy ending.

It is time to go back for a final visit with WealthGrid, where they have been working through exactly the transformation design that we explored in Chapter 2. The third time was the charm in terms of migration attempts: By applying the correct patterns in the proper order at the right times, the company was able to complete the first phase of the full transformation process. A transformation is not truly done until any remaining monoliths are re-architectured, a gradual process that will probably take another year or two to accomplish. The truly major milestone that WealthGrid has achieved, however, is that they've gained the ability, understanding, and awareness to do this as a natural course of business. The company is confident that they can continue to evolve; meanwhile, they can clearly see the results, and rewards, of all the progress they've made.

WealthGrid's senior executives took a great deal more care to be involved with creating strategy for this third attempt, but to their credit they also recognized when it was time to get out of the way and let the middle managers and engineers take over executing the transition details. Jenny, the program manager who had catalyzed the company to go cloud native in the first place, was named Transformation Champion in recognition of her initiative and the knowledge she and her team had gained through their first two attempts. Both these efforts failed, and in very different ways, but were valuable learning experiences nonetheless.

Jenny led the Core Team and helped choose its members. They were able to work off to one side in creative mode to find a transformation path that *would* work for WealthGrid, while the rest of the company's engineers focused on proficiently delivering features on the existing platform to keep customer value high in the meantime. Within three months, the Core Team was able to uncover the best approach for a new cloud native system through Exploratory Experiments and PoCs. At that point, a Platform Team was formed by splitting off several Core Team members; they were

joined by a few more engineers who had shown particular ability during the organization's failed "Let's let everyone build a platform and see what happens!" overinvestment in creativity during the first part of attempt number 2.

The Platform Team delivered a solid production-ready cloud native platform in under six months. Meanwhile, Jenny and the remaining Core Team members focused on developing some simple microservices applications both to test the new platform and create processes around developing on it. The Core Team was also preparing to onboard the remaining teams. They created excellent Dev Starter Pack materials, trainings, and Demo Apps while evangelizing the soon-to-come system across the organization, building excitement and buy-in. When the platform was ready, the development teams were incrementally onboarded onto the new system. Very quickly, WealthGrid was delivering new features and functionality for its products working entirely on the new cloud native platform (and using a newly evolved cloud-optimized delivery approach).

The entire company has gradually evolved to be working in the cloud native way: Build-Run Teams doing CI/CD and other cloud native methods to rapidly develop new features and deliver quick, incremental improvements—which the business teams are delighted to be included in planning. CEO Steve and the executive teams have embraced Dynamic Strategy. Throughout the entire organization there is a focus on delivery, but now there is also continued and meaningful investment in ongoing innovation. Even some pure research into emerging new technologies—Jenny has been named Designated Strategist and is now directing research, as well as shepherding innovation through all areas of the company. Everything is working well, and everyone is feeling confident. Once any remaining "Lift and Shift at the End" monoliths have been re-architected it will be time to shut down the old data centers for good. WealthGrid is well on its way to being a fully cloud native company.

What's Next?

The next steps for WealthGrid are the same next steps for any company completing a cloud native transformation: keeping the balance between proficiency and creativity that current conditions dictate. The entire story of WealthGrid's transformation journey can be shown as this fluctuating balance, graphed in Figure E-1, according to the different stages and the different ways they tried to reach the cloud.

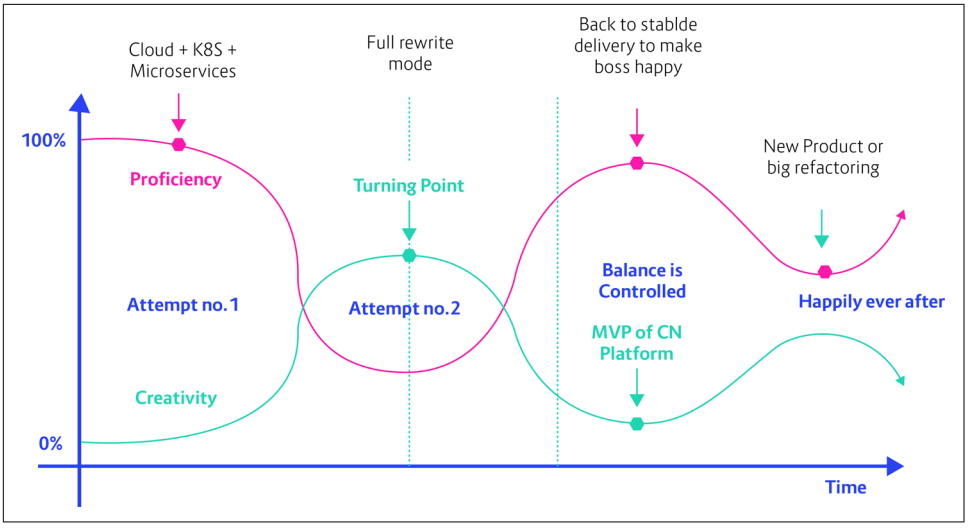


Figure E-1. The relationship between proficiency and creativity at different points along WealthGrid's transformation journey

Attempt number 1 is where WealthGrid does not treat the transformation as a full initiative and remains in primarily proficient mode while Jenny's small team tries to implement a new cloud native platform as a side project while also still responsible for their regular tasks. Proficiency is very high, and creativity almost zero.

This didn't work, so the company tried to swing the other way. There was Executive Commitment this time: a big budget allocated and most of the people moved to work on the initiative. Unfortunately, they were still using proficient management and Waterfall team structure—and still treating the transformation as a normal project that is supposed to have predictable results. Thus, the process and the culture did not change to match the new tech. Meanwhile there was so much investment in the new platform that the current platform—the one still in production and delivering the business value to the customers, not to mention earning money—was almost ignored. As a result, the numbers were out of whack again. They were working at perhaps 10 percent investment in delivery, while most of the rest of the company was experimenting all over the place. They were being creative, but in a very disorganized way that generated more problems than it solved.

Once WealthGrid finally stepped back, reassessed, and then came back for their third try, they had a much more balanced approach. By using a Dynamic Strategy, Gradually Raising the Stakes, and in general evolving their processes, management, and culture to be appropriately cloud native they were finally set up for success. A small portion of the engineers formed the Core Team and Platform Team to uncover and then build an MVP version of the new cloud native platform, so creativity swung

back to an effective but reasonable level while the remaining team members kept building the functionality customers needed.

At the end, with the new platform in place and all teams onboarded, WealthGrid could rebalance again for optimal efficiency: 80/15/5. Most of the focus (80 percent) returns to delivery, using the Manage for Proficiency approach with the majority of teams to efficiently deliver core business value. However, some teams remain in innovation mode, where Manage for Creativity keeps them effective at innovation—incorporating new tools and methods to significantly improve existing products—while a small percentage does continual research. Research means checking out new technologies that may or may not pan out, but it's important to understand how they work. Some of these may be useful in a year or two when they become more stable and/or more relevant for the business.

This is how WealthGrid handled things, but it works the same way for everyone. This balance is under your control. You can play with it as needed. You can decide to invest in a big refactoring or to build a new product and pull people into that project for the duration, temporarily placing a significant portion of your team on more creative and innovative tasks. During that time these people need to be managed in a different way, with more open-ended autonomy to create new things. Eventually, however, the problem is solved or the change is achieved, and things rebalance once more toward an emphasis on stable and efficient delivery. Until the next time.

Jedi Powers Unlocked

Ultimately this book, all these patterns, were never intended to teach you how to do a cloud native transformation (though it is certainly a helpful byproduct). We wrote this book to help you become comfortable with transformation as a process, so you can move forth confidently as an adaptive, responsive, aware organization ready for any stranger that comes your way.

You have now unlocked your power to deal with uncertainty—*any* uncertainty. That is the essential Jedi mind trick necessary for surviving and thriving in these ever-changing times. In the early days of software development, and up until very recently, everything was very segregated and competition was basically limited by geography. Now everything is fast and global: you are no longer competing only with your neighbors. You are competing with the world. To put it in disruptive stranger-danger terms, in the past you might encounter one stranger in your town every five to ten years. Now, you meet ten on the street every day.

Fundamentally, then, cloud native is *not* about speed, scale, or margin (though these are all terrific side benefits!). Yes, you can now vary your company's products or services in a cloud native pipeline with the same ease and speed you might update your website using a content management system. The point is not the speed itself, though,

but your ability to move at that speed. *Cloud native is about building an organization that can transform and adapt far faster by removing the technical risks associated with change.* With the tools we have outlined throughout this book, using these strategies and methods and patterns, you can handle any change that comes.

At its best, a cloud native approach is as much about de-risking as it is about accelerating change, allowing companies to become more responsive. It's extremely powerful, yes—but unfortunately "low risk" does not mean "low effort." Now that you know what you are doing, you still need to do the work. After reading this book you know how to structure your transformation, but you still have to go through a year or two of actually doing it. We can give you the knowledge and tools, but we can't save you the journey.

The transformation process isn't free, and it isn't easy. But what we can tell you is that it's imperative, in an existential threat sort of way, and absolutely worth it.

And the risk of undertaking it has never been lower.

About the Authors

Pini Reznik is CTO and cofounder of Container Solutions, a consultancy that is helping companies to successfully adopt cloud native technologies and practices.

In the five years of existence of Container Solutions, they've participated and led dozens of cloud native transformations and collected extensive hands-on experience in both technical and organizational aspects of the transformation.

Jamie Dobson is cofounder and CEO of Container Solutions, a professional services consultancy specializing in cloud migration. A first encounter with a BBC computer and BASIC at the age of nine launched a lifelong passion for programming and software development. He eventually developed a matching passion for coaching and organizational strategy to help humans work effectively and beneficially with the technology that increasingly drives our lives.

Michelle Gienow is a web developer, JAMstack evangelist, and former journalist whose clients include The New Stack, Linux Foundation, New York Times, and Discovery Channel, among many. She is happiest when working amid the fascinating confluence of technology and writing, whether it's JavaScript code or cloud native principles.

Colophon

The animal on the cover of *Cloud Native Transformation* is the Blue Clipper butterfly (*Parthenos sylvia*). It lives in the rainforests of India, Sri Lanka, Bangladesh, Myanmar, Papua New Guinea, and most of Southeast Asia. Its large (four inch) wingspan and beautiful colors wings make it a favorite of photographers and butterfly conservatories.

The name of the species means "virgin of the forest" in Greek. The English name refers to the 18th century clipper ships whose billowing sails the butterfly's white wing patches resemble. The wings' background color may be orange, green, or brown instead of blue.

The Blue Clipper is a fast, powerful flier, flapping its wings stiffly and keeping them mostly horizontal as well as gliding. The male is smaller than the female and flies closer to the ground, taking water from puddles. Both sexes also feed on nectar, often from lantana flowers. Larvae feed on tropical plants such as the passionflower and moonseed.

Many of the animals on O'Reilly covers are endangered; all of them are important to the world.

The cover illustration is by Karen Montgomery, based on a black and white engraving from *Encyclopedie D'Histoire Naturelle*. The cover fonts are Gilroy Semibold and Guardian Sans. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag's Ubuntu Mono.