**Chapter 8. Patterns for Organization and Culture**

Traditional organizations are structured to protect and reinforce the existing order: static strategy, top-down decision making, and little freedom for teams to deviate from the dictated path. This is in no way a bad thing! It is in fact an efficient structure for executing on a well-honed plan in a predictable and stable environment, but top-down flow takes a long time. This is fine if everything else is slow too, and releases happen only once every year or so.

Attempting to deliver on a new cloud native system by following the old ways is, however, a disaster in the making: things will break down very quickly because a hierarchical organizational structure simply can’t keep up with a cloud native delivery approach.

To succeed as a cloud native entity an organization must embrace freedom of choice, delegation, and individual independence and responsibility. Not because this is an inherently superior way to do things, but because it is the better way to support a dynamic strategy. The main power of cloud native is the ability to adjust whatever you are doing (or the way you are doing it) to adapt in response to market or other environmental changes. If change happens every few months—which it will—but requires coordinating a massive rollout through many layers of bureaucracy each time, this basically derails any strategic shift. By the time you finally get it out there the market (and, likely, some of your customers) will have moved on.

Becoming a cloud native organization requires more than just opening an Amazon Web Services account and installing Kubernetes. Your culture needs to evolve along with your tech (something we discussed in depth in [Chapter 2](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch02.html#the_human_challenge_of_cloud_native)). Culture is a consistent set of basic actions that add up to how people are encouraged to do their daily work. Having a cloud native culture means moving to be more proactive and more experimental, allowing your teams to be independent and to make their own decisions—and take their own risks. It means, essentially, deconstructing the waterfall.

This chapter collects a series of patterns to guide your evolution toward cloud native culture, to help you cut down dependencies and move toward decentralization. You are stepping out into an uncertain world and, yes, it’s a little scary. The old world was a well-charted, comfortable place. You knew how to get around, and also there were plenty of maps. This new landscape, though, changes all the time. Worse, there are few fixed landmarks—and no map at all. But you don’t need one. After all, the only way to find your way in this new territory is to explore it until you get your bearings and can navigate instinctively. Or, to frame it in terms of Simon Wardley’s seminal essays on optimizing organizational structure for continuous innovation, be a Pioneer rather than a Town Planner:[**1**](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#ch08fn1)

*Pioneers are brilliant people. They are able to explore never before discovered concepts, the uncharted land. They show you wonder, but they fail a lot. Half the time the thing doesn’t work properly. You wouldn’t trust what they build. They create ‘crazy’ ideas. Their type of innovation is what we call core research. They make future success possible…*

*…Town Planners are also brilliant people. They are able to take something and industrialise it taking advantage of economies of scale. This requires immense skill. You trust what they build. They find ways to make things faster, better, smaller, more efficient, more economic, and good enough. They build the services that Pioneers build upon.*

These patterns are the survival kit of tools and techniques that will help you forge your own particular trail as a true pioneer in this new world among the clouds.

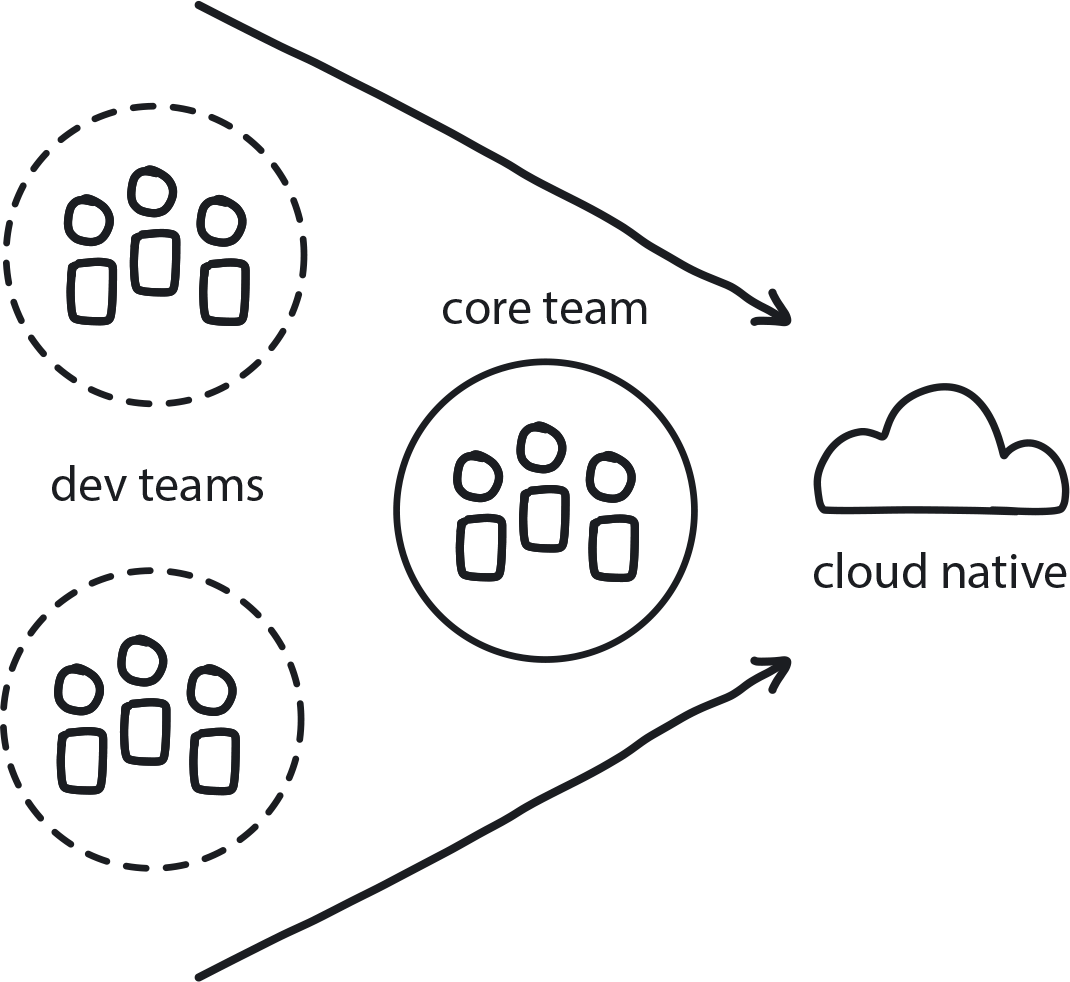
*The following patterns describe and address cloud native culture and organizational structure. They are presented in an order that we believe will be useful or helpful for the reader, but there is no right (or wrong) order for approaching them: Patterns are building blocks for a design and can be combined in different ways according to context.*

*This chapter is intended as an introduction to the patterns themselves, and there is intentionally little explanation relating them to each other at this point. When considering an individual pattern, the decision is not just where and when to apply it, but whether to apply it at all—not every pattern is going to apply in every transformation or organization. Once the concepts are introduced we will fit them together, in progressive order and in context, in*[*Chapter 11*](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch11.html#applying_the_patterns)*and*[*Chapter 12*](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch12.html#applying_the_patterns_a_cloud_native)*, as a design that demonstrates how patterns are applied in a typical cloud native transformation.*

* [Core Team](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#core_team)
* [Build-Run Teams (“Cloud Native DevOps”)](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#build_run_teams_devops)
* [Platform Team](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#platform_team)
* [SRE Team](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#sre_team)
* [Remote Teams](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#remote_teams)
* [Co-Located Teams](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#co_located_teams)
* [Communicate Through Tribes](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#communicate_through_tribes)
* [Manage for Creativity](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#manage_for_creativity)
* [Manage for Proficiency](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#manage_for_proficiency)
* [Strangle Monolithic Organization](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#strangle_monolithic_organization)
* [Gradual Onboarding](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#gradual_onboarding)
* [Design Thinking for Radical Innovation](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#design_thinking_for_radical)
* [Agile for New Development](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#agile_for_new_development)
* [Lean for Optimization](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#lean_for_optimization)
* [Internal Evangelism](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#internal_evangelism)
* [Ongoing Education](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#ongoing_education)
* [Exploratory Experiments](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#exploratory_experiments)
* [Proof of Concept (PoC)](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#proof_of_concept)
* [MVP Platform](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#mvp_platform)
* [Decide Closest to the Action](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#decide_closest_to_the_action)
* [Productive Feedback](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#productive_feedback)
* [Psychological Safety](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#psychological_safety)
* [Personalized Relationships for Co-Creation](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#personalized_relationships_for_co_creat)
* [Blameless Inquiry](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#blameless_inquiry)

**Pattern: Core Team**

*Dedicate a team of engineers and architects to the task of uncovering the best transformation path and implementing it along the way. This reduces risk embedded in the transformation while the team gains experience helpful for onboarding the remaining teams later (*[*Figure 8-1*](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#core_team01)*).*



**Figure 8-1. Core Team**

With Vision First in place, the company is now allocating resources to the cloud native transformation and choosing the best teams for leading the initial stages.

**In This Context**

**Making an existing team or teams responsible for delivering the new cloud native system while still requiring them to work on their regular duties means they will have conflicting priorities—and struggle to deliver either of them at all, much less do it well.**

This happens very commonly when companies fail to recognize the size and scope of a cloud native transformation—both the time and resources it will require and the many ways it will impact the organization. Instead, they treat it like another system upgrade added to the backlog of standard tasks. This leads to insufficient resource allocation to the cloud native transformation project and conflicting priorities for the engineers working on it and will almost certainly cause the initiative to stall or simply fail.

* Teams working on both urgent and important tasks will tend to prioritize urgent tasks first, leading to deprioritization of important tasks such as cloud native transformation.
* Cloud native technologies are new and complex. They require intense time investment for learning and experimentation.
* Some of the cloud native challenges are too difficult for one person to handle.
* A team responsible (and trusted) for delivering a new solution will have full commitment to the solution and later evangelize about it across the organization.

**Therefore**

**Create a single Core Team of five to eight engineers and architects to lead the transformation.**

* Team responsibilities will include ownership of the technical vision and architecture and derisking the transformation by running a series of experiments to uncover the best transformation path.
* Later, the Core Team can use its experience to onboard other teams to the new way of working and to create the Dev Starter Pack materials.
* The team may continue improving the platform after the major parts of the transformation are done.

**Consequently**

**The Core Team rapidly works through the most challenging parts of the transformation (identifying the best migration path, tools and technologies, and then implementing a minimum viable product version of a platform) and paves the way for the rest of the teams in the company toward successful cloud native adoption.**

* + The team is building knowledge and experience in the cloud native area.
* + It can use this growing knowledge to adjust the vision and the architecture of the applications as it goes.
* + Progress is visible and measurable.

**Common Pitfalls**

Underestimating the complexity of cloud native work and treating it as just another technical upgrade. See [Chapter 9](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch09.html#patterns_for_development_and_process).

**Related Biases**

*Planning fallacy*

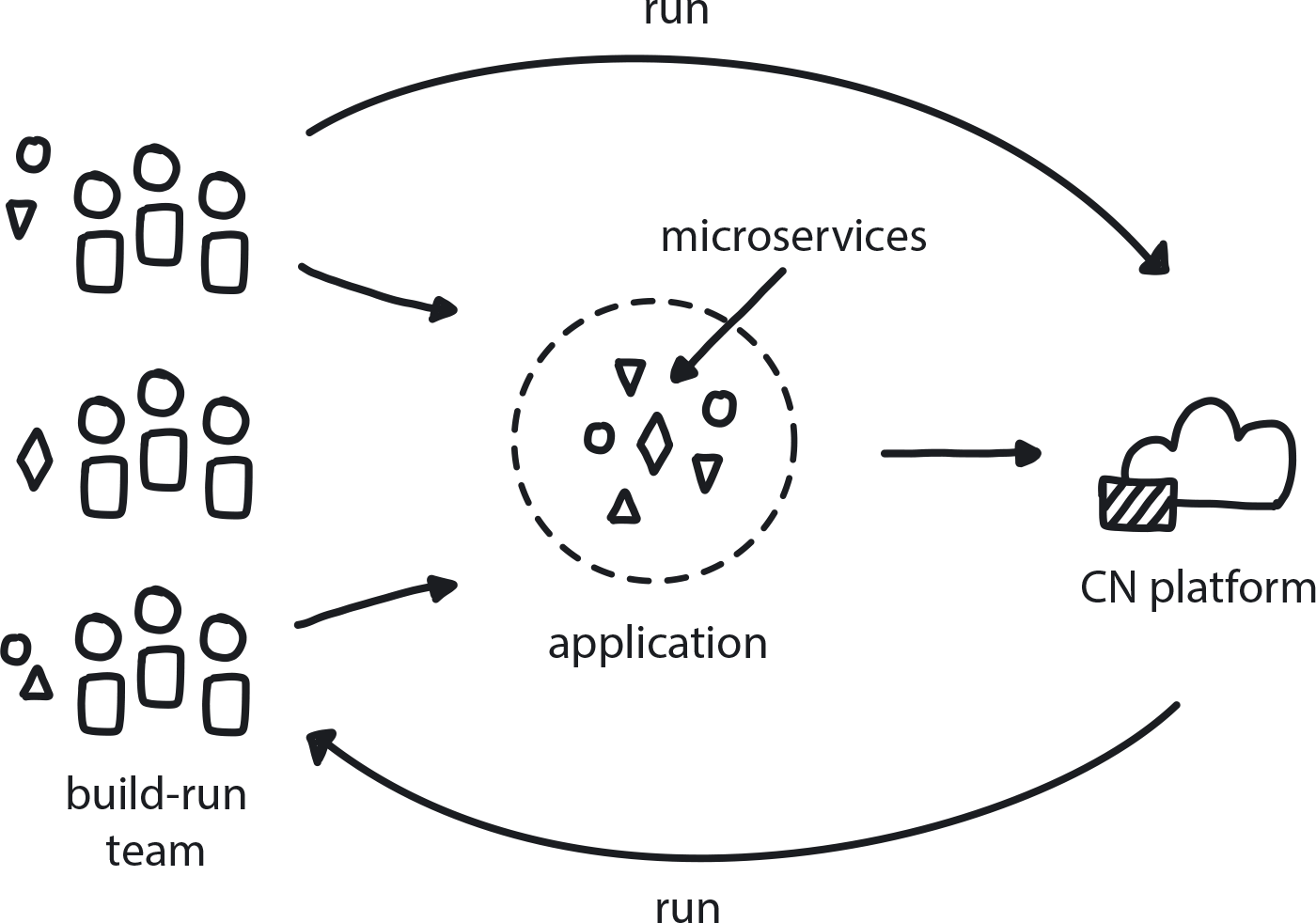
The tendency to underestimate task-completion times.

**Related Patterns**

* [Vision First](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#vision_first)
* [Gradually Raising the Stakes](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#gradually_raising_the_stakes)
* [Reference Architecture](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch09.html#reference_architecture)
* [Demo Apps](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch09.html#demo_applications)
* [Build-Run Teams](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#build_run_teams)
* [Avoid Reinventing the Wheel](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch09.html#avoid_reinventing_the_wheel)
* [Gradual Onboarding](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#gradual_onboarding)
* [Developer Starter Pack](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch09.html#developer_starter_pack)

**Pattern: Build-Run Teams (“Cloud Native DevOps”)**

*Dev teams have full authority over the services they build, not only creating but also deploying and supporting them (*[*Figure 8-2*](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#build_run_teams)*).*



**Figure 8-2. Build-Run Teams**

The company has cross-functional teams (Agile) or teams siloed by technical specialty (Waterfall) and needs to move to a structure compatible with cloud native. Development teams rely on the Ops team to deploy artifacts to production. The company is looking for the right balance between independence and standardization for their dev teams.

**In This Context**

**When development teams are responsible for building an application and supporting it in production, if they also try to build the build the platform to run it, the organization can end up with multiple irreconcilable platforms. This is unnecessary, expensive to operate (if even possible), and takes time away from teams that should be focusing on delivering features, not the platform they run on.**

* Handover between development and operations teams kills production speed and agility.
* Adding an Ops person on each developer team is how you end up with 10 irreconcilable platforms.
* Conway’s law says that software architecture will come to resemble the organization’s structure, so if we want the platform to be independent, then the teams developing an application need to be separate from the team running the production platform.
* There is a limit to the capabilities a team can have. Devs are devs; they can extend their knowledge to a certain level, but they are not Ops.
* Maintaining Ops and Development as separate disciplines/teams is not sustainable in cloud native.
* Too much freedom brings chaos, but too much standardization strangles innovation.

**Therefore**

**Create teams that each have their own capability to build a distributed system with microservices managed by dynamic scheduling.**

Build-Run teams will be responsible for building distributed applications, deploying them, and then supporting the applications once running. Build-Run teams all use the same standardized set of platform services and deploy to a single unified platform that runs all applications for the entire company. This platform is the responsibility of the Platform Team, which implements and supports it.

* Build-Run teams are not DevOps teams in the traditional sense that devs and operations people sit together.
* In Agile, the development team also includes software testing capabilities, but the products are handed over to a separate Ops team to be delivered to production.
* In cloud native a true cross-functional team must be able to build distributed systems.
* The Platform Team is a specific kind of Build-Run team in that it builds, deploys, provisions, and supports the cloud native platform and infrastructure, but it works separately from the application development teams.

**Consequently**

**There is strong separation of defined responsibilities: Build-Run teams handle the applications. The Platform Team is responsible for building and maintaining the operating platform.**

The Platform Team doesn’t have to be part of the company—public clouds like Google/AWS/Azure, etc., with their automated platforms, could make an internal Platform Team unnecessary. If a Platform Team is designated, they are typically global, supporting all services/applications; Build-Run teams are separate and rely on standardized platform delivered by the Platform Team.

* + Teams have a bounded level of autonomy and ability to focus on their true tasks.
* + Developers still have freedom to choose components to run on top of the standardized platform so long as they are compatible.

**Common Pitfalls**

Giving Build-Run teams the responsibility for creating their own platforms. They deploy Kubernetes for their service, in their own way; but then there are 10 other teams creating their own platforms for their own services in *their* own way.

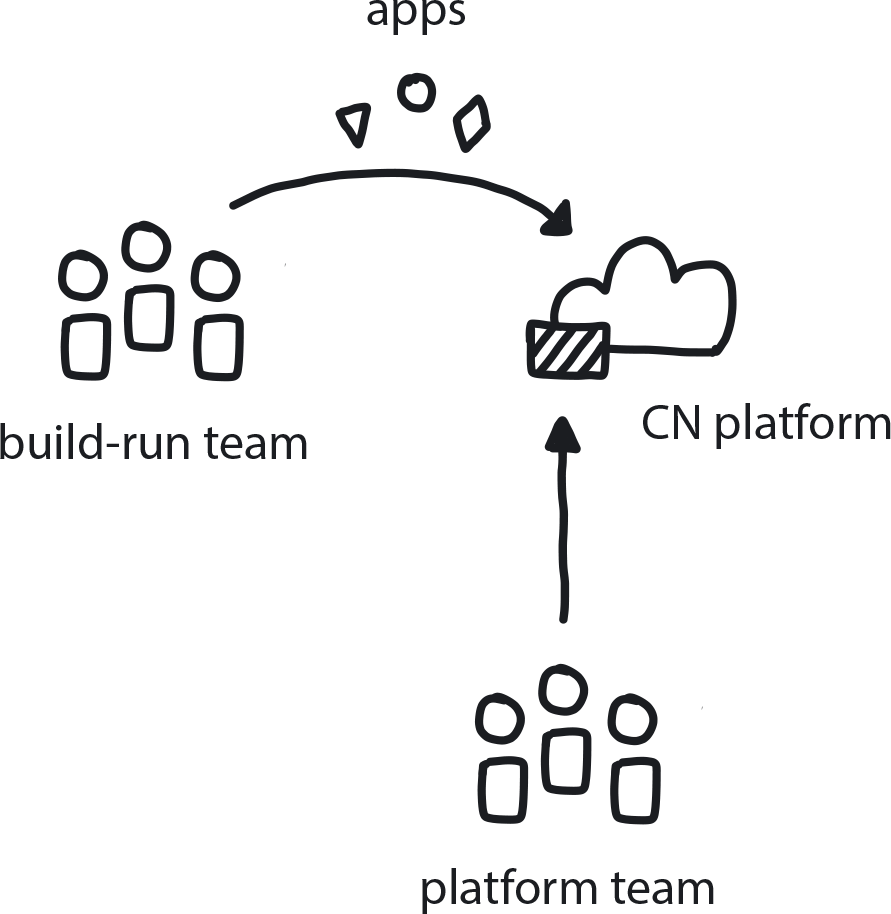
Adding Ops engineers to development teams (basic DevOps principle) instead of having a designated Platform Team (see pattern). This leads to each team having its own platform, producing a multitude of competing cloud native platforms that are difficult to develop and coordinate, and expensive to maintain.

**Related Patterns**

* [Platform Team](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#platform_team)
* [SRE Team](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#sre_team)
* [Avoid Reinventing the Wheel](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch09.html#avoid_reinventing_the_wheel)
* [Public Cloud](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch10.html#public_cloud)

**Pattern: Platform Team**

*Create a team to be in charge of architecting, building, and running a single, consistent, and stable cloud native platform for use by the entire organization so that developers can focus on building applications instead of configuring infrastructure (*[*Figure 8-3*](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#platform_team01)*).*



**Figure 8-3. Platform Team**

An enterprise is moving to the cloud and adopting microservices architecture. Many teams are building many different services, and they need extra tools to create the infrastructure to run their pieces of the application.

**In This Context**

**If there is no single team in charge of creating an official cloud native production platform for the transformation, each team responsible for different microservices will have to build its own platform.**

This duplicates effort, wastes time, and—most critically—sows conflict when it comes time to deploy. Because each service was built using a different approach on a bespoke platform, they will have conflicting needs and be difficult, if not impossible, to run all on one unified platform. Ops is stuck trying to come up with some kind of Frankenstein solution, but it’s unlikely they will be able to make it work.

* Different teams will build different solutions if not coordinated.
* Basic cloud is not enough; complex and custom configuration is required.
* Standardization maximizes the possibility of reuse.
* Freedom of choice is needed for finding the best solutions.
* People are creative in finding ways to circumvent the roadblock set by lack of platform, leading to a “shadow IT” effect.

**Therefore**

**The Platform Team will handle the platform itself—everything between the cloud and the apps (or, to put this in terms of the current technological landscape, Kubernetes and below)—while developers are responsible only for building the applications themselves (again, in the current tech landscape, Kubernetes and above).**

* Set up a separate team responsible for choosing, building, and configuring a basic set of tools and practices to be used by all of the development teams.
* Build one platform to be used by the entire organization.
* Create a standard and reusable set of tools to simplify work for the devs.

**Consequently**

**Developers are able to focus on building individual application services, features, and functionality while the Platform Team takes care of running the platform itself. Developers are allowed to introduce custom tools but they will have to support them as part of their own application unless the tools are proven stable and/or requested by other development teams.**

* + There is a consistent, reusable, and, above all, stable platform that ensures all services work together seamlessly.
* + There is less work for developers: they can just focus on building apps and not worry about the platform they will run on.
* − There is also less freedom for developers to choose their tools, although choice is still possible.

**Common Pitfalls**

The Platform Team forgets about its main goal of building a good and useful platform for the development teams and instead sets off to build the best ever, most scalable, and most technically amazing platform with availability of 99.999999%. Why? Because they can.

Imagine your company wants to build a nice three-story office building, but the architect comes back with a plan for a 100-story skyscraper. Constructing such a massive edifice would obviously be a crazy waste of time and money, right? Well, the same is happening with cloud native platforms.

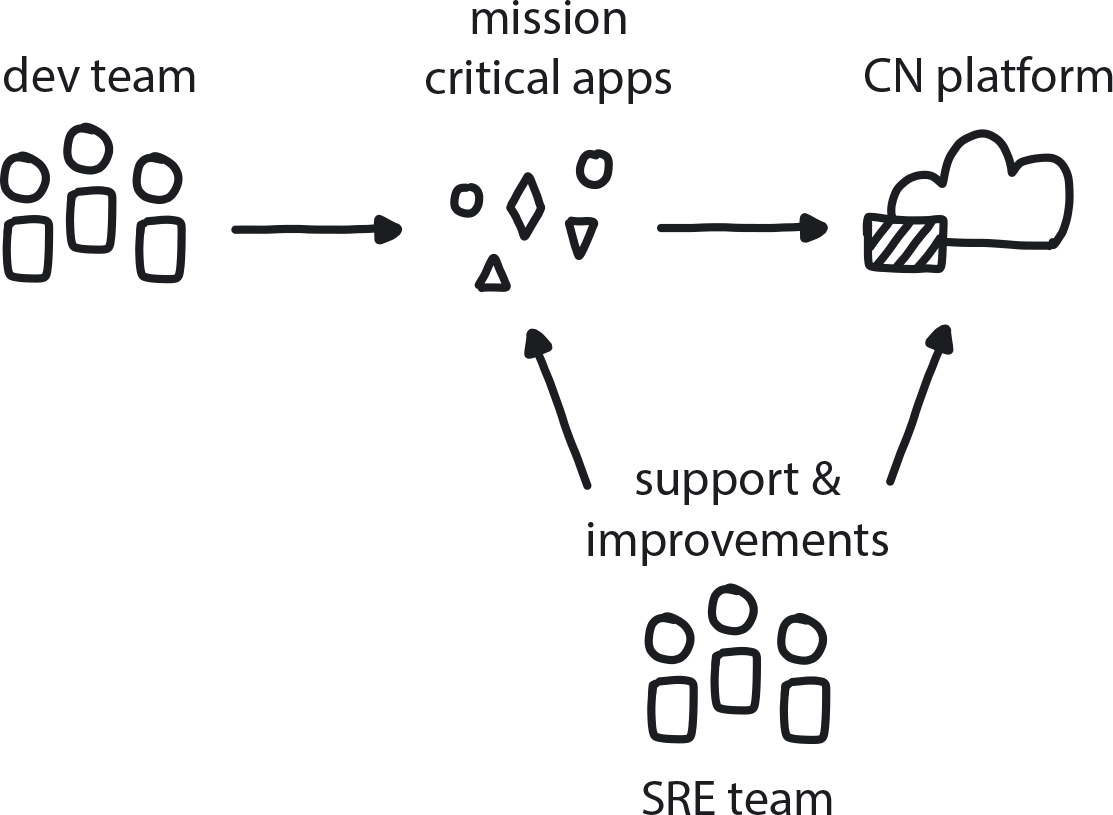
Such projects get approved because managers don’t fully understand the technology and fear confronting the head architect head-on. In most cases, early versions of the platform can be quite basic because the applications that are initially going to run on it are themselves quite basic. The quality and scalability of the platform should be built gradually as both the development teams and their applications mature and grow, and their actual platform requirements emerge more clearly.

**Related Patterns**

* [Core Team](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#core_team)
* [Build-Run Teams](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#build_run_teams_devops)
* [Reference Architecture](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch09.html#reference_architecture)

**Pattern: SRE Team**

*The SRE (Site Reliability Engineering) team helps the development teams to maintain and improve the application (not the platform or infrastructure) (*[*Figure 8-4*](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#sre_team01)*).*



**Figure 8-4. SRE Team**

A big company has a large, mission-critical application with very high demands for quality and availability, and significant resources for creating dedicated improvement teams.

**In This Context**

**Once a platform is built and in production, attention is often directed away from improving internal processes and runtime performance. This can cause degradation over time, reducing quality and performance.**

* Dev teams are measured on functionality, Ops on stability, SRE on improvements.
* SRE is a very expensive team to operate because it requires the most experienced and knowledgeable engineers.
* Opportunity cost arises when you pull your best engineers from development teams to SRE.
* If improvements are the priority for this team, then this is somewhat removed as a responsibility/priority for the Build-Run teams.
* SRE is more relevant when an application enters maintenance mode, rather than during the initial build.

**Therefore**

**Create a team that is focused 50% on reliability and 50% on continuous improvement of internal processes and development practices.**

This SRE team worries about overall site (platform) availability. However, each individual service has its own operational needs as well. It can be helpful to add SREs into each individual build squad (or at least tribe) to focus on service availability.

* SRE engineers are in charge of improving whatever it takes to create better infrastructure, better runtime, and better user support.
* The SRE team makes the error budgets and helps the dev team define its operational model and the Platform Team to improve the platform.
* Most tasks are automation rather than manual support.
* Every 18 months the SRE team needs to automate everything it is doing manually (so it can move to the Platform Team’s responsibility).
* You have a highly knowledgeable and experienced team, with good knowledge of operations but also devs who can write code.

**Consequently**

**The runtime stability and quality is continuously increasing, and automation is also increasing.**

* + Developers are aware of operational concerns and incorporate that knowledge into their development cycle.
* + The SRE team works closely with the Build-Run teams and the Platform Team.
* − SRE teams are expensive and take top engineering talent away from other projects.

**Common Pitfalls**

Taking your best five engineers away from other work to try to capture the last 1% of performance gain.

Having an SRE team when the application has no actual need for this kind of improvement, instead of using that brainpower to build products. It’s expensive in both money and opportunity cost, but people will do it anyway because they think they need an SRE team because other companies have them (Bandwagon effect). In such companies, SRE will typically not live up to the expectations and will forever remain a normal Ops team, only with a fancier title.

**Related Biases**

*Bystander bias*

If no one is in charge of improving internal processes, improvement is not likely to happen. Conversely, if the SRE team is in charge of improvements, that takes some responsibility off Build-Run teams to optimize their contributions.

*Bandwagon effect*

Everyone is talking about doing Kubernetes, so we’d better do it too!

*Status quo bias*

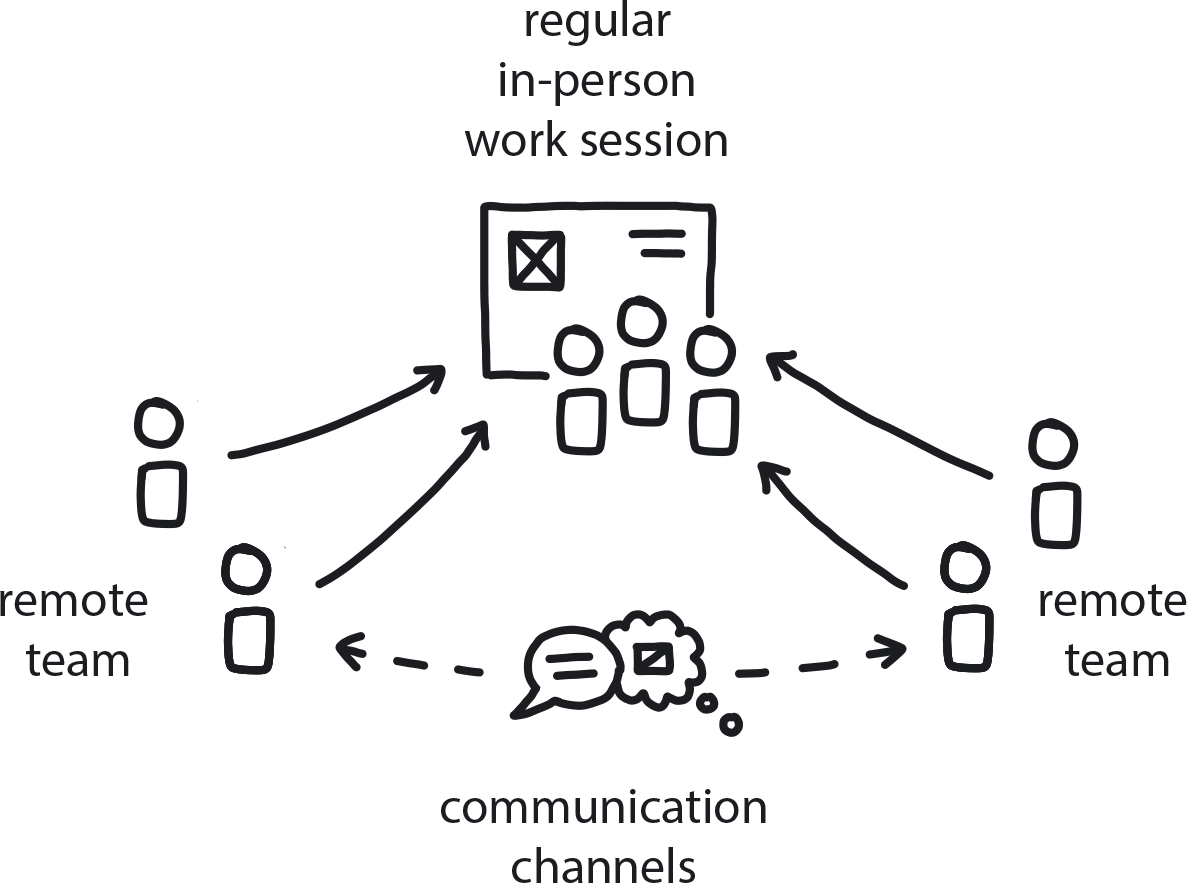
It’s working just fine, so let’s just leave it alone (and changing something that works feels risky).

**Related Patterns**

* [Build-Run Teams](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#build_run_teams_devops)
* [Platform Team](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#platform_team)
* [Delayed Automation](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch09.html#delayed_automation)

**Pattern: Remote Teams**

*If teams must be distributed, whether across a city or a continent, build in regular in-person retreats/work sessions as well as robust channels for close and free-flowing communication (*[*Figure 8-5*](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#remote_teams01)*).*



**Figure 8-5. Remote Teams**

A company is solving complex and relatively difficult business problems, and many team members are working remotely from one another.

**In This Context**

**In many organizations remote team members may rarely, or even never, meet face to face. That works as long as the problems being solved by those teams are reasonably well-defined and not very complex. In the complex world of cloud native, however, problems are often messy and difficult, and require a more open-ended and collaborative approach.**

**Without a strong aim for collaborative co-creation, the team’s ability to generate innovative solutions is typically limited to the creative abilities of individual team members working separately.**

* Distributed workforces are increasingly common, and a team can have members scattered across a country, a continent, or the entire globe.
* The cloud native approach rests upon teams being given goals to achieve along with the freedom to decide how to execute them.
* In a predictable environment, little communication is required for executing stable workloads.
* Problem-solving without talking to anyone else means you are solving the problem separately, not collaboratively.

**Therefore**

**Put programs in place to connect remote teams and bring them together in every way possible, both physically and virtually.**

In the complex world of cloud native, problems are often messy and difficult, and require a more open-ended and collaborative approach. Working well this way requires in-person relationships, honest feedback, and close communication. If teams must be scattered, then regular face-to-face meeting times and other ways to connect the individual members need to be prioritized in order for them to be an effective cloud native workforce.

* Hold regularly scheduled in-person team retreats/gatherings/offsite work sessions.
* Open multiple channels of communication using tools like Slack to keep teams in constant and fluent contact.
* Make use of other tools like video conferencing, remote whiteboarding, and virtual *fika* sessions (see below) to keep members connected and working closely.
* In the end, there is no real substitute for personal interaction, sharing a meal, etc.
* If possible, don’t distribute a team until it is already mature and has created strong internal connections.

**Consequently**

**Teams see each other regularly in person and in between stay engaged via multiple channels and practices that promote fluent communication. Ideas are created, validated, and implemented in groups instead of coming from individuals.**

* + Increased productivity and innovation are a natural outcome of closer ties with co-collaborators.
* + Regularly scheduled events—both daily hangout sessions and quarterly meetups—are easy to plan around so work is not disrupted.
* − Offsite retreats, though important, are an additional operating expense.

**Common Pitfalls**

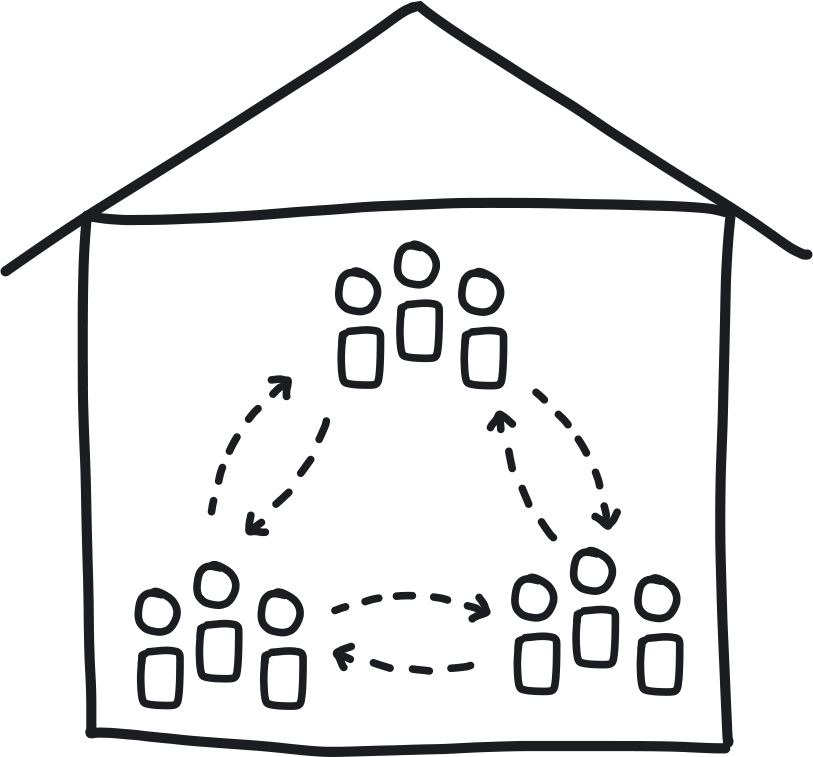
Confusing time spent sitting at one’s desk with productivity. Swedish culture has a tradition known as *fika*. While it roughly translates to drinking coffee, in the workplace *fika* is a scheduled opportunity to connect and bond with others. To some work-obsessed cultures, stepping away from your desk to mingle with coworkers may sound like a waste of billable workday time units. However, recent data from the Organization for Economic Cooperation and Development suggests that Sweden’s higher productivity rates per hours worked, compared with countries known for their long work hours such as the United States, Japan, and Korea, are at least in part due to *fika*.

**Related Patterns**

* [Co-Located Teams](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#co_located_teams)
* [Personalized Relationships for Co-Creation](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#personalized_relationships_for_co_creat)
* [Communicate Through Tribes](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#communicate_through_tribes)

**Pattern: Co-Located Teams**

*Teams that work together in person develop naturally closer relationships and better collaborative problem-solving abilities, which in turn nurtures greater innovation (*[*Figure 8-6*](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#co_located_teams01)*).*



**Figure 8-6. Co-located Teams**

An organization that is moving to cloud native has multiple offices and office buildings. Teams that are targeted to move to cloud native have individual members located in multiple physical locations.

**In This Context**

**When team members are located in different places, they tend to communicate less and focus on their jobs rather than personal relationships. This hobbles team problem-solving, because individuals will attempt to solve problems separately and then contribute that solution back to the team—rather than solving them collaboratively with teammates.**

* Creative teams especially are more effective when personal relationships are on a higher level.
* Complex communication is better in person.
* Cloud native is a paradigm shift that requires new ways of working and creative problem-solving.

**Therefore**

**All members of a given dev team will work in the same physical location and meet daily.**

All meetings and collaboration will be first done using whiteboards, pen and paper, etc. Teams will be chosen based on location. If the team is in the same office, then move the members to sit at adjacent desks.

* Periodic team-building events to strengthen personal relationships.
* Encourage the team to talk to one another in person.
* Consider pair programming.

**Consequently**

**High level of trust and proximity naturally increases collaboration.**

* + Quick and effective communication.
* − Not always possible due to distribution of teams.
* − Possibility of personal conflict exists.

**Common Pitfalls**

Assuming that just because people sit together, they are a team and will be effective working together. Team-building efforts are still required—see *Personalized Relationships for Co-Creation* for specific ways to create a bonded team.

**Related Biases**

*Bandwagon effect*

The tendency to do (or believe) things because many other people do (or believe) the same thing can be a hazard in groups.

*Bystander effect*

The tendency to think that others will act in an emergency situation translates, in the team context, to no one volunteering to take responsibility for a task unless it officially “belongs” to them.

*Curse of knowledge*

A group of people sharing the same knowledge and experience tends to forget that others outside the group don’t have access to that same knowledge base.

*Shared information bias*

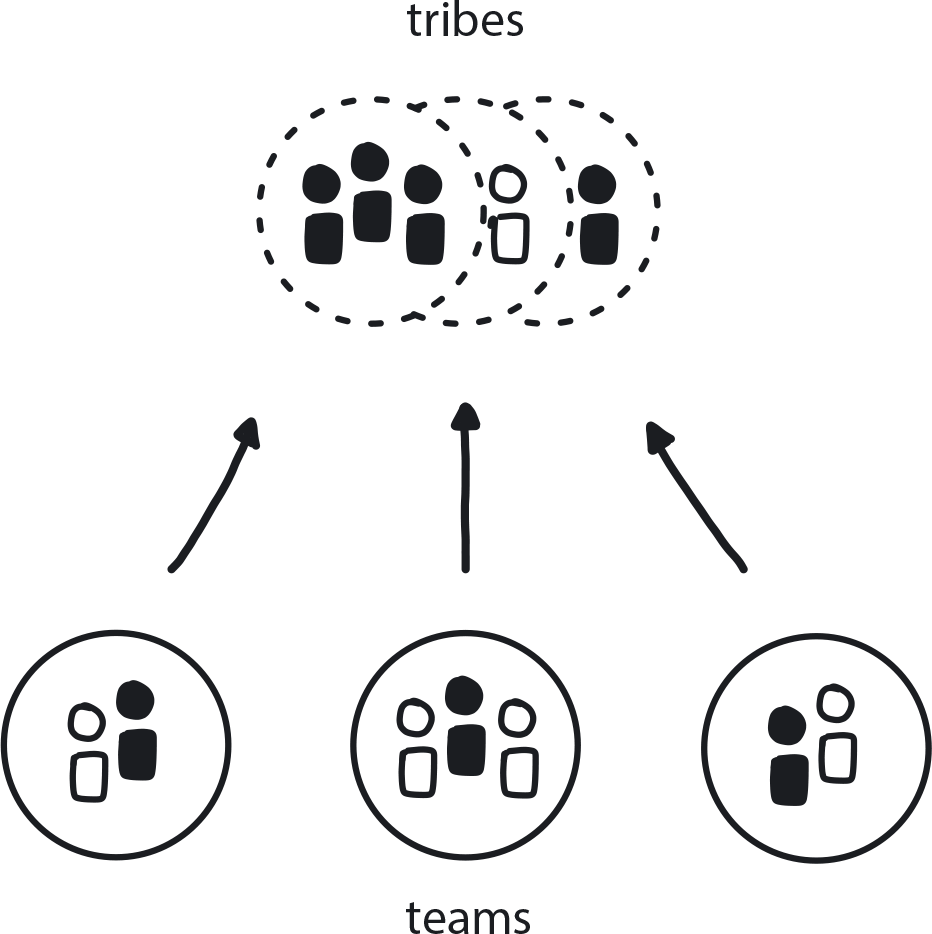
Group members are likely to spend more time and energy discussing information that all members are already familiar with (i.e., shared information). For example, the whole team did Docker training, so they spend a lot of time talking about Docker—and no time at all about Kubernetes, which is equally necessary but they don’t know much about it. To counter this, especially in new and complex environments, teams should learn new things all the time.

**Related Patterns**

* [Personalized Relationships for Co-Creation](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#personalized_relationships_for_co_creat)
* [Remote Teams](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#remote_teams)
* [Manage for Creativity](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#manage_for_creativity)
* [Ongoing Education](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#ongoing_education)
* [Communicate Through Tribes](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#communicate_through_tribes)

**Pattern: Communicate Through Tribes**

*Create groups of people who have similar skills but are on different teams to cross-pollinate ideas across the company and provide valuable whole-organization perspective (*[*Figure 8-7*](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#communicate_through_tribes01)*).*



**Figure 8-7. Communicate Through Tribes**

A company is moving from a hierarchical Waterfall approach to the quickly evolving and complex world of cloud native. In traditional organizations, decision making and knowledge sharing are done according to hierarchy, and managers or lead architects are responsible for knowing everything about their assigned projects. In cloud native, however, engineers are given ownership over microservices and need to make decisions quickly and independently. Delivery processes are being fully automated.

**In This Context**

**In a changing cloud native world, with ownership for application services divided across teams, managers don’t know enough to provide effective advice, much less make good decisions. At the same time, managers have the illusion of knowledge and control—they don’t know what they don’t know—so the team’s abilities and effectiveness are only as great as its manager’s capability.**

* Cloud native tech is complex and takes time to learn.
* Most delivery decisions in cloud native happen during automated delivery process or very close to it.
* Dependencies between teams and a need to get permission for any change slow down delivery.

**Therefore**

**Create domain-specific tribes that operate outside of normal management hierarchy.**

Use tribes to share specific technical and organizational info and to provide perspective and advice for complex decision making.

* Members of tribes belong to different teams.
* Meetings are regular and open-ended.
* Tribes play advisory and coordination roles but have no decision-making power.
* All the tribe members are knowledgeable in the tribe’s domain.

**Consequently**

**The company has groups that cross-cut traditional organizational units. This helps those people who are closest to and most knowledgeable in a particular domain subject identify areas for running experiments and making changes.**

* + Tribe members share ideas and advise one another on issues and problems.
* + Managers have limited ability to intervene.

**Common Pitfalls**

Tribe meetings need to be interactive and discussion-oriented, but too many turn into “Here’s some lunch while you watch this slide talk” sessions. This type of passive format stifles communication and the free exchange of ideas, which is the entire point of these meetings.

**Related Biases**

*Curse of knowledge*

A group of people sharing the same knowledge and experience tends to forget that others outside the group don’t have access to that same knowledge base.

*Shared information bias*

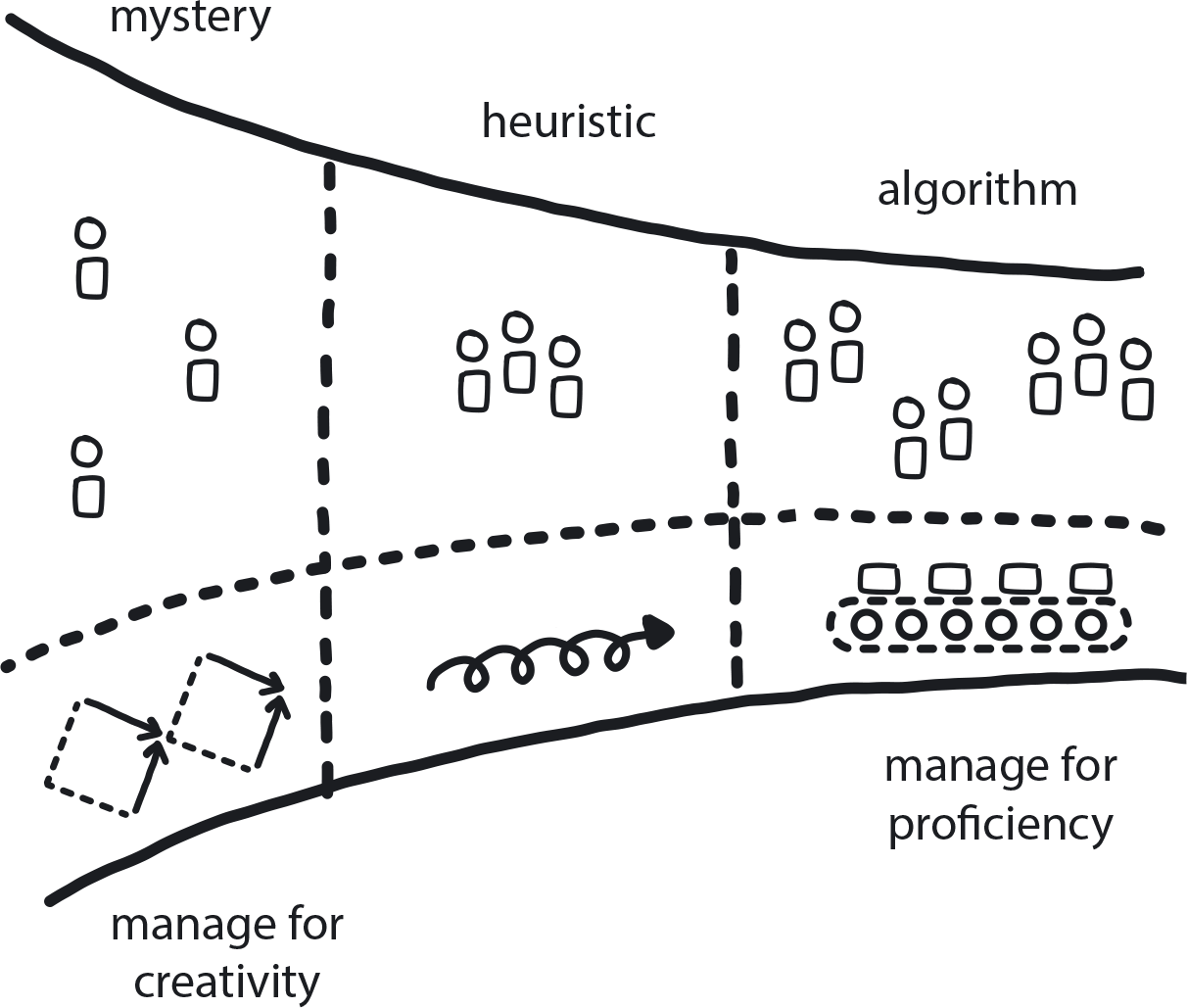
Group members are likely to spend more time and energy discussing information that all members are already familiar with (i.e., shared information). For example, the whole team did Docker training, so they spend a lot of time talking about Docker—and no time at all about Kubernetes, which is equally necessary but they don’t know much about it. To counter this, especially in new and complex environments, teams should learn new things all the time.

**Related Patterns**

* [Co-Located Teams](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#co_located_teams)
* [Remote Teams](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#remote_teams)
* [Personalized Relationships for Co-Creation](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#personalized_relationships_for_co_creat)

**Pattern: 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 (*[*Figure 8-8*](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#manage_for_creativity01)*).*



**Figure 8-8. Manage for Creativity**

The company is investing in all Three Horizons—H1, proficiency; H2, innovation; and H3, research. Some teams are working to deliver H1 core value and improve proficiency. Other teams are dedicated to innovation projects, which require a creative approach.

**In This Context**

**Teams charged with identifying and building promising future products are often managed using the same known methodologies that are popular in the enterprises. One of the most common, Scrum, helps clarify what’s going to be built and creates pressure to build it as fast as possible. Running endless numbers of strings without much reflection on the way drives most of the creativity out of the development project.**

**No inventor can tell when and what exactly she is going to invent.**

* Creative thinking requires time and freedom to explore, and safety to fail.
* Startups tend to manage for creativity.
* Enterprises tend to strongly promote proficiency, speed, and quality of delivery.
* Scrum and similar methodologies create pressure to deliver and reduce free thinking.

**Therefore**

**Manage the teams responsible for innovation by stating a purpose or desired outcome, which gives the team a direction toward which they will be creating new ideas. The team will require time, funding, and other resources to support its work, safety to fail, and autonomy to explore. Team dynamics will be more important than deadlines and delivery management.**

Cloud native companies need creative thinkers. This new tech-driven environment requires companies to be innovative, flexible, and responsive. While many teams within the organization can be focused on delivering the core business product, at least one should be in charge of innovation. Because its job is to investigate likely next steps for the company’s near future, it needs to work differently than the teams executing processes in the existing system.

* Psychological safety is essential for creativity to flourish.
* Purpose needs to be practical and achievable.
* Creative teams should be dedicated to innovation and have no regular delivery tasks.
* Innovation Champions are responsible for ushering projects that prove to be of value toward production, moving them along among the H1/H2/H3 teams. This role can be covered by the Designated Strategist or another person who can be officially responsible.

**Consequently**

**Innovation thrives in the company, and the innovative teams are separated from the proficient teams.**

* + Creative teams have the opportunity to invent and the safety to fail.
* − No certainty in delivery, but the projects are always time-bound.
* − Difficult to predict results looking forward, but easy to measure success looking back.

**Common Pitfalls**

Taking creativity too far: digital transformation is the balance between innovation and pragmatism. When faced with the pressure to evolve due to some new disruptive competitor, some companies go all in on innovation and get lost. They miss the point of balance: you need to be *more* creative, not *all* creative. For example, Google is one of the most innovative companies in the world, but it has 2% creativity and 98% delivery. Companies that want to be like Google almost always get this backward, thinking Google is 98% creative, when in fact it is laser-focused on optimum delivery of its services.

Trying to manage creative teams to deliver in a proficient way, i.e., specific deliverables on a set schedule. This creates pressure to deliver and reduces the ability to think creatively. Creative teams under proficient management work hard to produce the illusion of innovation by performing many different tasks, but they rarely create any meaningful breakthroughs.

**Related Biases**

*Shared information bias*

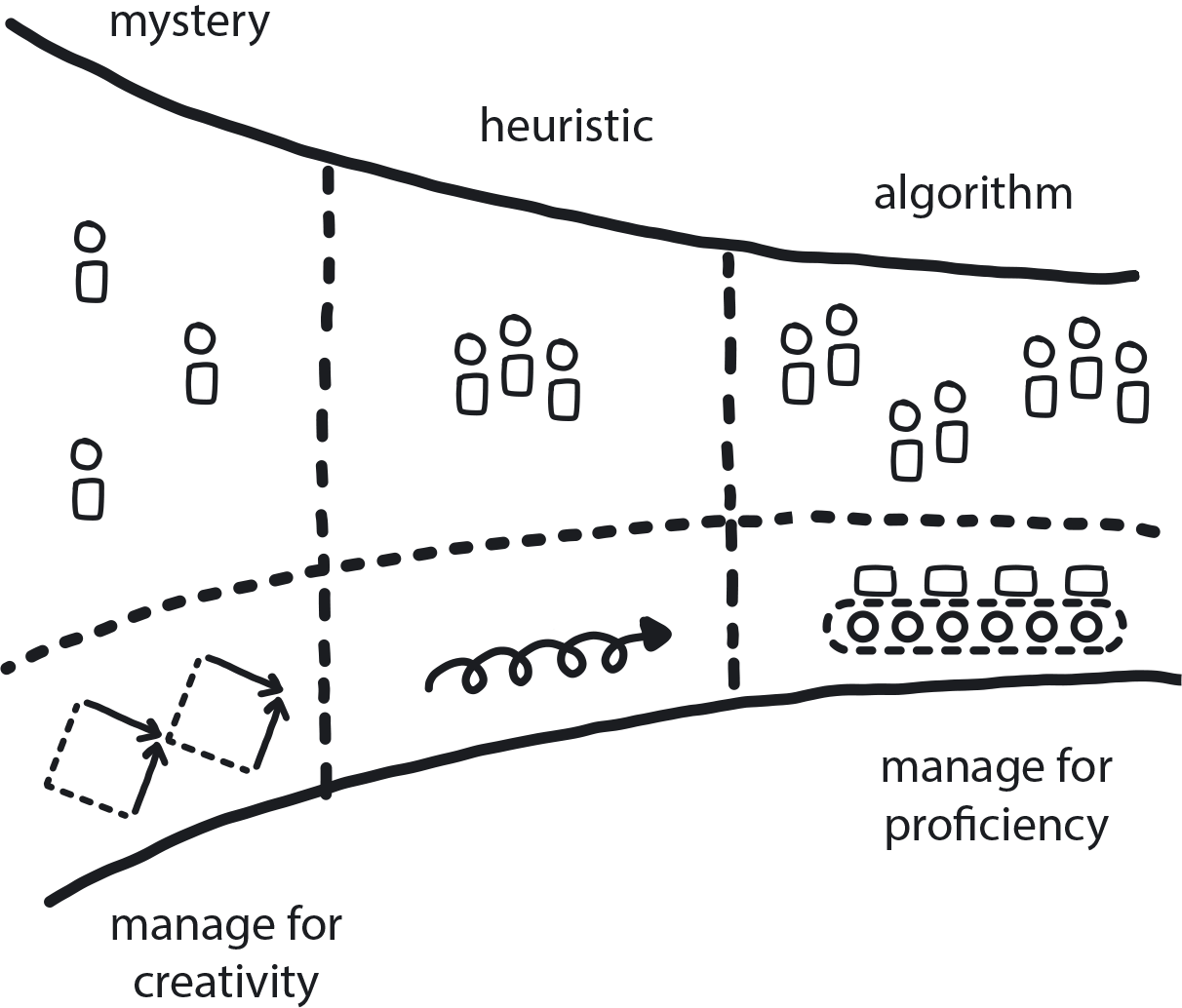
If an innovation team is functioning under delivery pressure (that is, being managed for proficiency instead of creativity), it risks trying to speed things up by focusing on things it already knows about. In this context it can feel like there is no time for branching outward into new or untried possibilities.

**Related Patterns**

* [Three Horizons](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#three_horizons_pattern)
* [Manage for Proficiency](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#manage_for_proficiency)
* [Learning Organization](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#learning_organization)
* [Reduce Cost of Experimentation](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#reduce_cost_of_experimentation)

**Pattern: Manage for Proficiency**

*Teams delivering stable and highly repetitive or algorithmic work should be managed for high quality and optimal efficiency (*[*Figure 8-9*](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#manage_for_proficiency01)*).*



**Figure 8-9. Manage for Proficiency**

A transformation is underway. Some teams are working creatively to build the new system while other teams remain focused on keeping the existing system delivering the company’s core products/services.

Alternatively, a cloud native transformation is at its end, and all the teams onboarded to the platform know very well what they are doing. They are ready to deliver excellent products to customers as fast as needed.

**In This Context**

**When the teams responsible for delivering stable and scalable products are given too much freedom to innovate and explore, they introduce new risks that harm product quality and increase the cost of development.**

**In many cases the new cloud native platform is not yet ready or stable enough to accommodate all the new teams, while most of the teams have to maintain old systems and release new incremental updates. Allowing those teams to start innovating too early may come at a significant cost to productivity and product quality.**

* Established companies desire creativity because usually they don’t have enough of it.
* Startups are highly creative yet desire proficiency because they are striving to build a consistent, deliverable product.
* The typical enterprise culture strongly supports a proficient, even algorithmic, way of working.

**Therefore**

**Run the execution teams the way they have always been run. Focus on repeatability and optimize on quality and speed of delivery.**

As an organization transforms itself into a cloud native entity, most teams will eventually need to evolve to cloud native’s more flexible, intuitive, and problem-solving way of working. During the transition, however, there will be teams still tasked with delivery of the core business on the existing system while the new platform is built. 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.

Once a system (or new product) has left the innovation phase and is in steady production, most resources can typically be moved away from research and experimentation, and back to a mostly proficient approach.

* Let them know that they are valued: keeping existing product lines going at high quality is important to the business.
* Promise them that they will be onboarded to the new system when the time is right.
* Aim to create team pride around the best possible results and highest-quality products.
* Management of proficient teams requires high repetition, high feedback, and a strong sense of belonging.
* Emphasize that both types of teams are needed and important to the organization; avoid the perception or reality that creative work is rewarded more.

**Consequently**

**Teams in charge of delivering the company’s profit-generating products/services in a proficient way are being managed to optimize this. Proficient and creative teams are loved and appreciated equally.**

* + If the market changes in a large and/or frequent way, the company can shift resources into creativity and innovation when necessary. But creative is expensive, so drop back to proficient when things are stable/predictable.
* − Some teams may need to work in the legacy codebase permanently (see “Lift and Shift at the End”).

**Common Pitfalls**

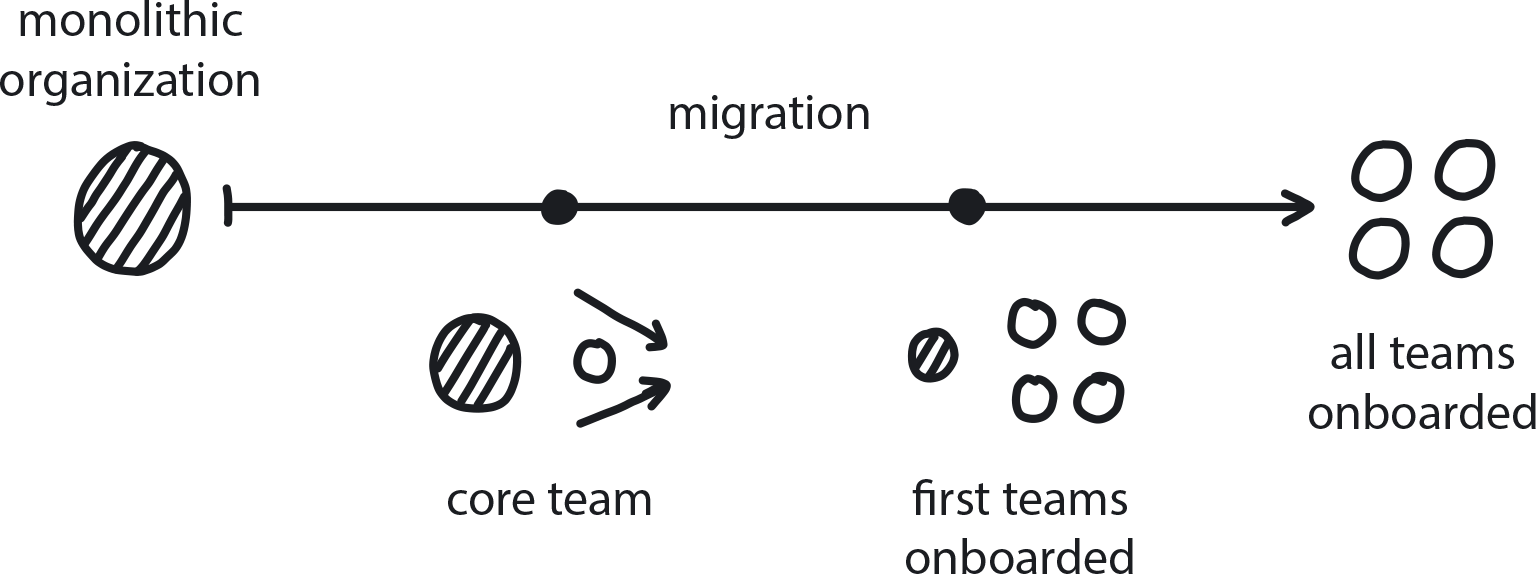
Trying to train all the teams at the same time in the new cloud native tools and methods. Delays before a team is onboarded to the new system because it is still running the old one can sow frustration or, worse, fear among its members that they could be made redundant, or resentment that they are being left out. Also, training well before team members have a chance to practice new skills typically is a waste, and the training will need to be repeated later anyway. Keep managing these teams for the old proficient system until the time is right for them to move over.

**Related Patterns**

* [Manage for Creativity](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#manage_for_creativity)
* [Lift and Shift at the End](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch10.html#lift_and_shift_at_the_end)
* [Strangle Monolithic Organization](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#strangle_monolithic_organization)

**Pattern: Strangle Monolithic Organizations**

*Just as the new tools, technologies, and infrastructure gradually roll out over the course of a transformation initiative, the organization and its teams must also evolve to work with them properly (*[*Figure 8-10*](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#strangle_monolithic_organizations01)*).*



**Figure 8-10. Strangle Monolithic Organizations**

A cloud native transformation is in progress and some teams are moving to cloud native, while others may not move to cloud native for a long time.

**In This Context**

**Migrating an existing company to the cloud can take years, and it happens very gradually.**

There are two types of moves that rarely work. First, trying to move everyone over at once does not lend itself to good training. Without a solid background in the new tech, teams will be less effective and get frustrated over time or fall back on old habits that don’t work well in cloud native. If the cultural and organizational change is not supported to evolve equally along with the cloud native technology, this also creates frustration.

Second, simply leaving one part of the organization in legacy while everyone else moves to cloud native stratifies the company so that the teams that never get moved to cloud native are stalled professionally. They’ll never get to play with the cool new tech and build modern developer skills if they are left behind to babysit the legacy system. This, naturally, leads to frustration, resentment, and reduced motivation, not to mention difficulty hiring and retaining engineers.

* Cloud native transformation takes a long time.
* When people learn something, they need to apply it very soon.
* Tech and org/culture misalignment leads to frustration.
* People are motivated by hope of future improvements or frustrated by lack of hope.

**Therefore**

**Move the teams from the legacy organizational structure to the new one gradually (Gradual Onboarding pattern). Restructure teams and change from hierarchy shortly before the new onboarding to the cloud native platform when it is fully ready.**

This is an organizational version of Martin Fowler’s architectural strangler pattern: Slowly strangle the process, and Waterfall cultural relics like specialized teams, along with the monolith itself. The two systems, old and new, can coexist well during the migration process. Once complete, the remaining pieces of the legacy system can be ported over to the new cloud native platform (Lift and Shift at the End) and gradually refactored into microservices until the old monolith is no more, and the maintainers are now experienced with cloud native.

* Educate constantly.
* Promote experimentation.
* Shift from hierarchy to tribes and delegation.
* Avoid training and restructuring if the team is not planning to move to cloud native soon.
* Create a plan for all legacy teams but execute only when the move is close.

**Consequently**

**The old system keeps working as always while the new one is built, and teams are gradually moved over. Teams get restructured and retrained only when it is time for them to actually move. While you are on the old platform you keep delivering with excellence; then you move to the new one and deliver equally well there.**

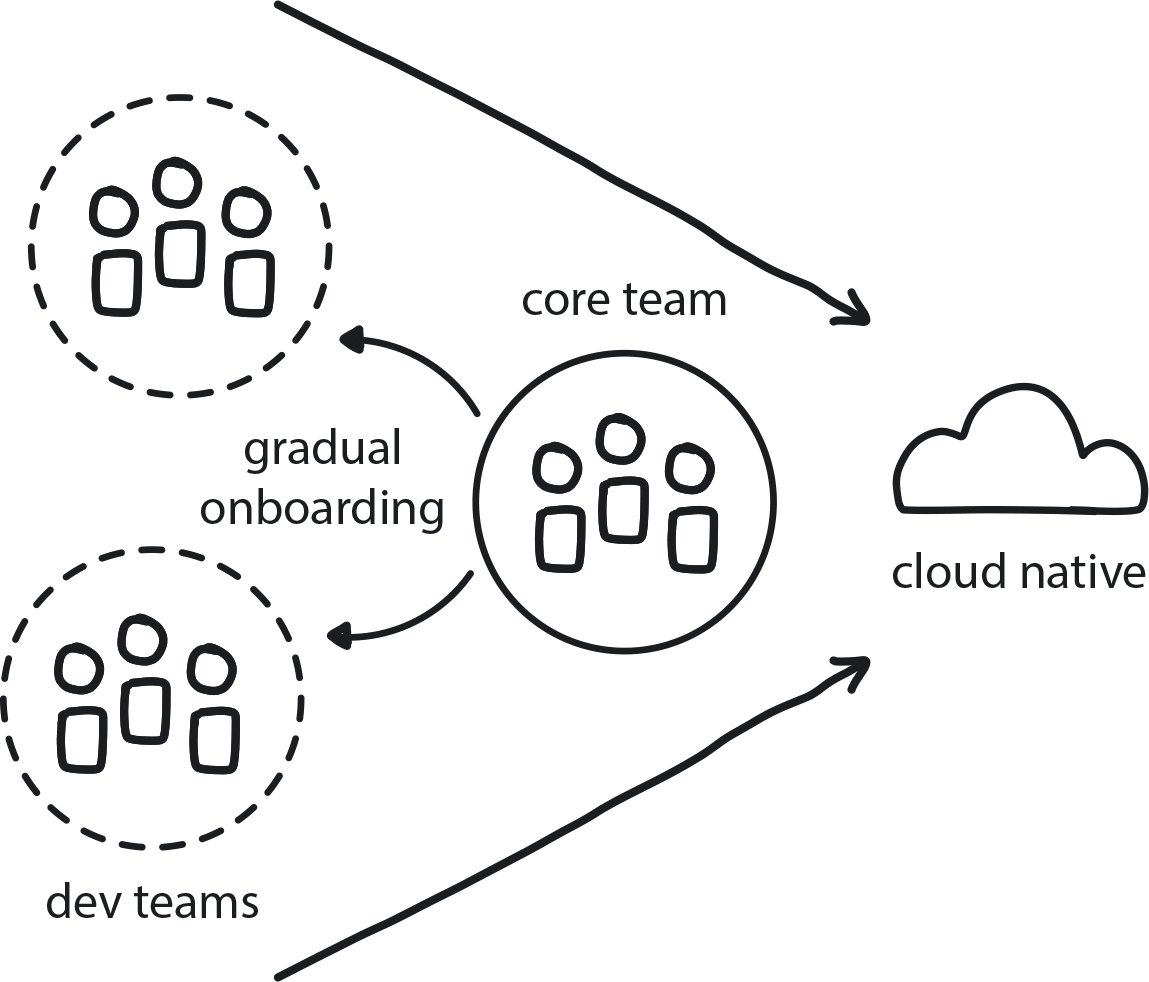
* + There is a clear plan for all the teams.
* + Organizational/cultural changes are aligned with tech changes.
* + Original and new cultures are coexisting but only temporarily until the full cloud native transfer is complete.
* − Potential clashes between teams.
* − Legacy teams could be disappointed by lack of change.

**Related Patterns**

* [Decide Closest to the Action](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#decide_closest_to_the_action)
* [Communicate Through Tribes](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#communicate_through_tribes)
* [Lift and Shift at the End](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch10.html#lift_and_shift_at_the_end)
* [Gradual Onboarding](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#gradual_onboarding)
* [Manage for Proficiency](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#manage_for_proficiency)
* [Ongoing Education](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#ongoing_education)
* [Internal Evangelism](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#internal_evangelism)

**Pattern: Gradual Onboarding**

*One to three months before the new platform goes live, begin training a couple of teams at a time with a pause between each cohort to incorporate feedback and improve the process/materials (*[*Figure 8-11*](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#gradual_onboarding01)*).*



**Figure 8-11. Gradual Onboarding**

There is pressure from the top, the executives and board, to start using the cloud native platform as soon as possible and pressure from below, from the development teams, to move to cool technology. The company spent a lot of time and money building the new system, and everyone is eager to reap the rewards. Developers want to learn and use new technology.

**In This Context**

**Onboarding too many teams at once will stress the Core Team and reduce its ability to continue improving the platform. Educating people too early, however, will create anxiety in the teams and desire to start as soon as possible (and frustration if there is a long wait before the new system is available).**

* A team that releases a tool on a large scale with low quality will struggle to improve it.
* People have little patience and are eager to apply recent learning.
* If you learn something new, you need to start using that knowledge soon or risk losing it.
* Good people may go to other companies to have a chance to work with the new and exciting technologies.

**Therefore**

**Start small organizational changes early in the cloud native transformation. Prepare materials to onboard other teams and execute it slowly when teams are ready.**

**Continue onboarding the rest of the organization gradually over a period of 3 to 12 months.**

* Start educating teams one to three months before onboarding.
* Prepare Developer Starter Packs and all relevant educational materials to simplify the onboarding.
* Onboard two to five teams at a time. Take breaks in between to use their feedback to improve the platform.
* Continuously improve the platform and the onboarding materials based on the feedback from the recently onboarded teams.

**Consequently**

**The Core Team can support teams as they onboard and improve the process as they go. The first few teams onboarded to the platform can help educate and support the teams onboarded later.**

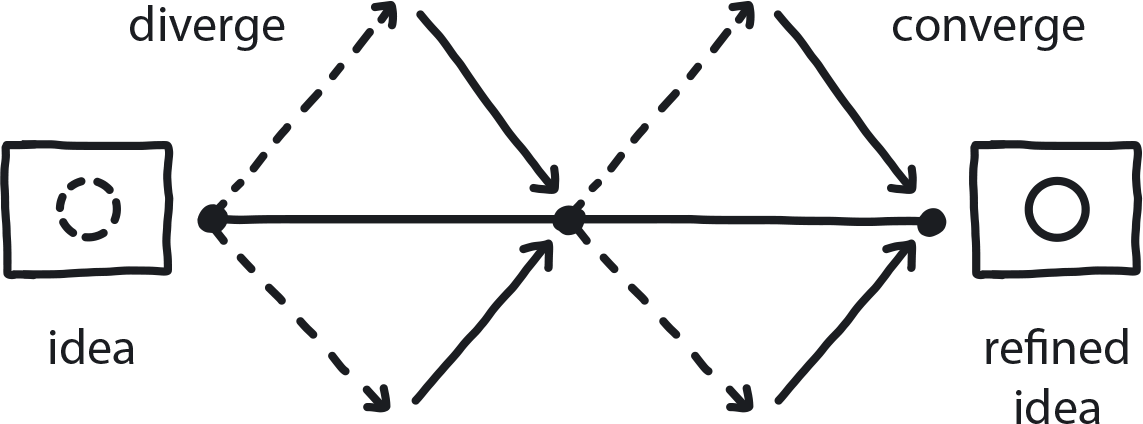
* − Some teams may not join for one to three years.
* − Slower perception of scale.

**Related Patterns**

* [Core Team](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#core_team)
* [Developer Starter Pack](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch09.html#developer_starter_pack)
* [Ongoing Education](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#ongoing_education)
* [Reference Architecture](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch09.html#reference_architecture)

**Pattern: Design Thinking for Radical Innovation**

*Whether faced with a radical new idea or a big problem, Design Thinking can be used as a process for first brainstorming a robust list of solutions and then narrowing it down to the best possibilities for actual exploration (*[*Figure 8-12*](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#design_thinking_for_radica)*).*



**Figure 8-12. Design Thinking for Radical Innovation**

We have a big idea or a difficult problem that may change the business. There are possible solutions, but also many uncertainties.

**In This Context**

**When faced with a problem, people typically spend only the minimum time required to find the first satisfactory solution, even if it’s not the best one or doesn’t involve all stakeholders. This leads to reasonable solutions, but misses the opportunity to find excellent ones.**

* Law of preservation of cognitive energy leads people to choose the first good solution.
* Developers typically are decision makers on tech solutions.
* Devs aren’t trained in business or market forces.
* Elegance and beauty are also relevant for internal systems.

**Therefore**

**Take the basic first idea and run it through a series of divergent and convergent thinking exercises to rethink the idea’s boundaries and explore alternatives.**

Ideation/brainstorming is followed by application experiments. In-depth explanation of Design Thinking is beyond the scope of this pattern, but guidelines for running Design Thinking workshops can be found online[**2**](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#idm46079287615560)

* Create 10 to 20 new solutions/problems/ideas based on the first idea, narrow that list to the two to three best ones. Then expand and iterate these two to three best ones into 10 to 20 new possible variations, and then again choose two to three of the most promising to run longer experiments and choose to implement.
* Hold regular Design Thinking workshops.
* Use the workshops to refine big ideas/problems.
* Invite major stakeholders.
* Facilitate collaboration to help excellent solutions emerge.

**Consequently**

**Ideas are thoroughly explored. Cost of initial exploration is still low, as it requires little to no actual development (No Regret Moves).**

* − Too many people involved slows down the decision-making process.

**Common Pitfalls**

People without experience in running Design Thinking workshops may end up running them more like Scrum planning exercises. Instead of helping with a wider exploration and evaluation of new and often radical ideas, it reaffirms the existing leading solution and effectively becomes an execution-planning session. A correctly led Design Thinking workshop does not have to lead to a new solution, but its main purpose is to question the assumption of the problem and brainstorm a variety of possible solutions.

**Related Biases**

*Ambiguity effect*

The tendency to avoid options for which missing information makes the probability of the outcome seem “unknown.”

*Confirmation bias.*

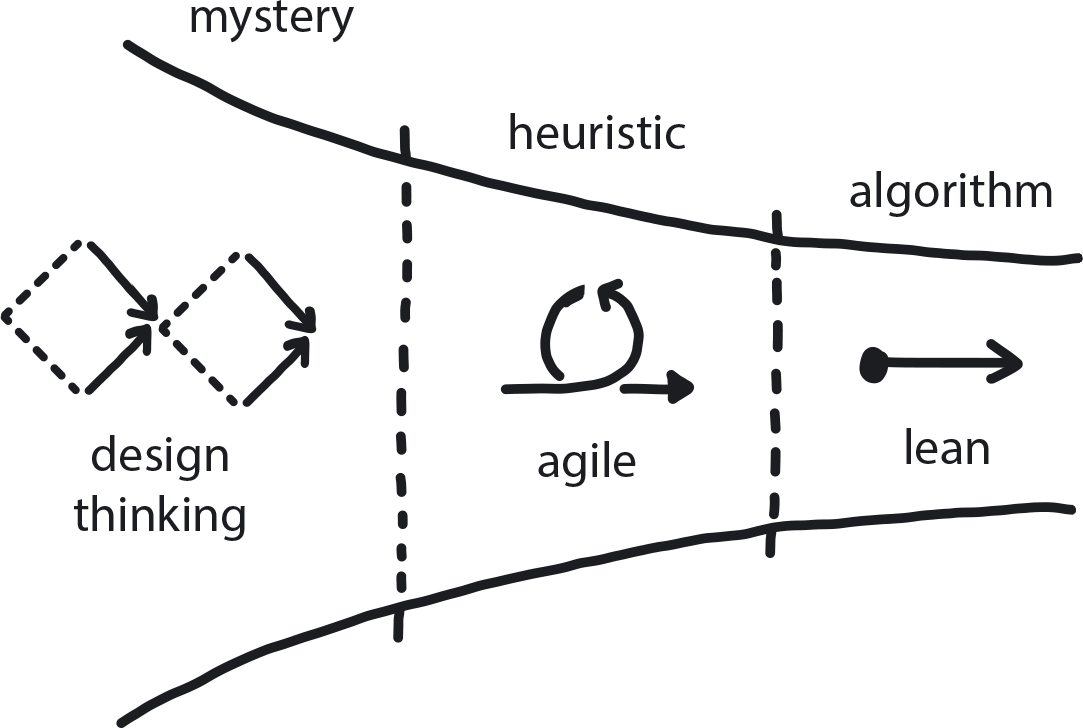
Picking test variables that will “prove” your existing opinion, rather than genuinely exploring alternatives.

**Related Patterns**

* [Involve the Business](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#involve_the_business)
* [Vision First](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#vision_first)
* [Exploratory Experiments](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#exploratory_experiments)
* [PoCs](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#proof_of_concept)
* [Reduce Costs of Experimentation](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#reduce_cost_of_experimentation)

**Pattern: Agile for New Development (Innovation Breaks)**

*Balance proficiency and innovation by building a separate time for each into your development cycle (*[*Figure 8-13*](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#agile_for_new_development01)*).*



**Figure 8-13. Agile for New Development**

Strategy is defined and solution direction is chosen after experimentation. There are still many uncertainties and not all knowledge is available within the team.

**In This Context**

**Teams are either endlessly researching and collecting information or, conversely, starting to deliver and optimize very early. In the first case, value is delivered to customers late, ends up being of poor quality, or never gets delivered at all. In the second case, the solutions are too simple and underdeveloped and miss the opportunity to solve customer problems.**

* Many processes aiming to increase productivity attempt to do so by pushing out innovation.
* Radical innovation and proficient delivery can’t be done in the same way.
* Innovation without delivery is academic research.
* Delivery without innovation is blind to market changes.

**Therefore**

**Run alternating iterations of research and development.**

Spend about 20%–30% of the time running proof-of-concept experiments and other research activities. Use the rest of the time to implement the discovered solutions and increase the quality.

* Every third or fourth sprint in Scrum should be used for research.
* 20%–30% of Kanban backlog should be dedicated to innovation and improvement tasks.
* When a solution is not clear, use experiments or PoCs to discover and eliminate options.

**Consequently**

**Delivery and innovation are separate and in balance.**

* + Team can still change product direction relatively easily.
* + Quality of the products constantly increases.
* − One-third of the time is not immediately monetized.

**Common Pitfalls**

Using Scrum by the book without leaving any space for PoCs and innovation. This leads to team exhaustion and to overly high focus on delivery too early in product development.

**Related Biases**

*Status quo bias*

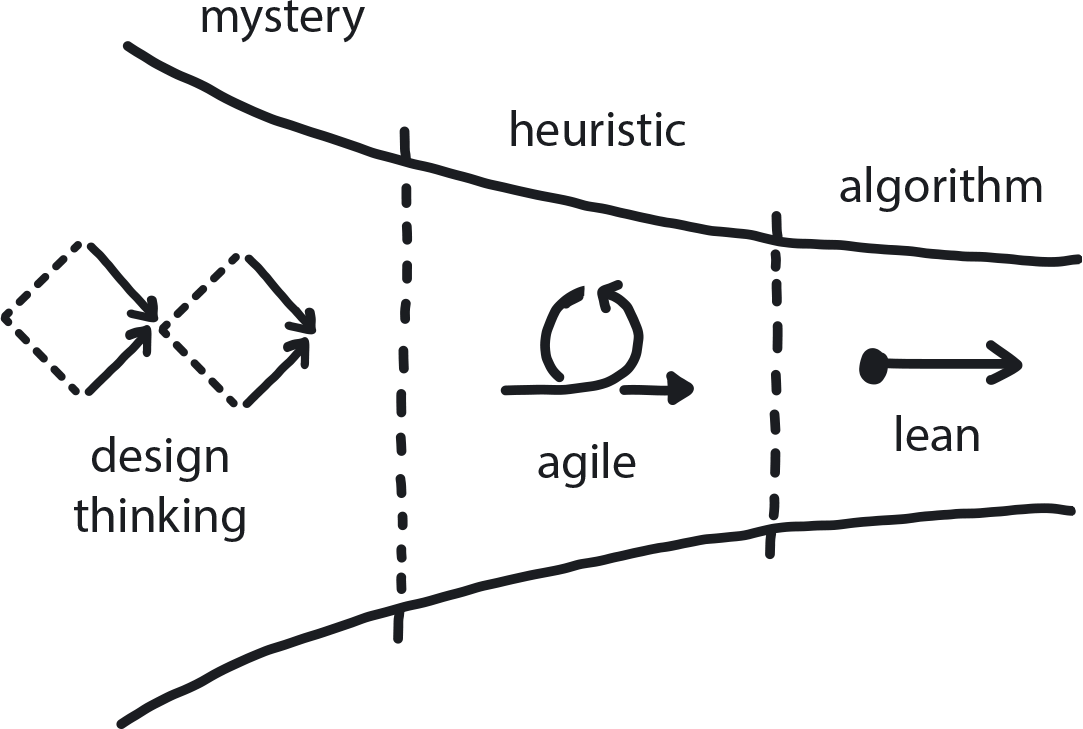
It’s always worked to do it this way before, so let’s just keep doing that. It might not be optimal, but we know it works (and changing something that works feels risky).

**Related Patterns**

* [Three Horizons](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#three_horizons_pattern)
* [PoCs](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#proof_of_concept)
* [Exploratory Experiments](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#exploratory_experiments)
* [Dynamic Strategy](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#dynamic_strategy)
* [Manage for Creativity](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#manage_for_creativity)
* [Reflective Breaks](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#reflective_breaks)
* [Gradually Raising the Stakes](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#gradually_raising_the_stakes)

**Pattern: Lean for Optimization**

*When a stable system delivers the value that’s intended and is not a target for technical innovation, focus on improving the system by continuously and incrementally improving delivery and maintenance processes with emphasis on repeatability (*[*Figure 8-14*](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#lean_for_optimization01)*).*



**Figure 8-14. Lean for Optimization**

Your product is stable and in demand (a “Cash Cow” on the BCG growth-share matrix), and it is in a state of incremental improvement and support. Tech used in the product is known and well-understood. All capabilities are present in the team.

**In This Context**

**Innovation and evolution are inevitable in technology. However, there is little need to innovate when a proficient system is delivering stable value and maintenance cost is low. And we often see that, in an otherwise proficient system, the team continues to introduce new tools and solutions that constantly destabilize the product while needlessly consuming time and budget.**

* Repeatability leads to boredom.
* Quality demands investment.
* Creativity and proficiency require different management frameworks.
* Proficiency and mastery require repeatability and standards.
* Creativity requires freedom and psychological safety for generating experimentation.

**Therefore**

**Reduce work in progress, focus on optimizing delivery process, measure quality and speed of delivery, and aim to improve both.**

Lean management optimizes proficiency and value to customers while eliminating anything that does not bring value to the end product/service and ensuring continuous incremental improvement.

* Use Kanban with optimization and new features backlog.
* Measure everything that matters.
* Optimize price versus performance ratio.
* Optimize proficient processes.
* Automate repeatable tasks.

**Consequently**

**Delivery is fast and proficient. System is stable, and quality is consistently going up.**

* + No destruction of an efficient system.
* + Gradually increasing quality.
* − Very limited innovation.
* − Some people are bored.

**Related Biases**

*Pro-innovation bias*

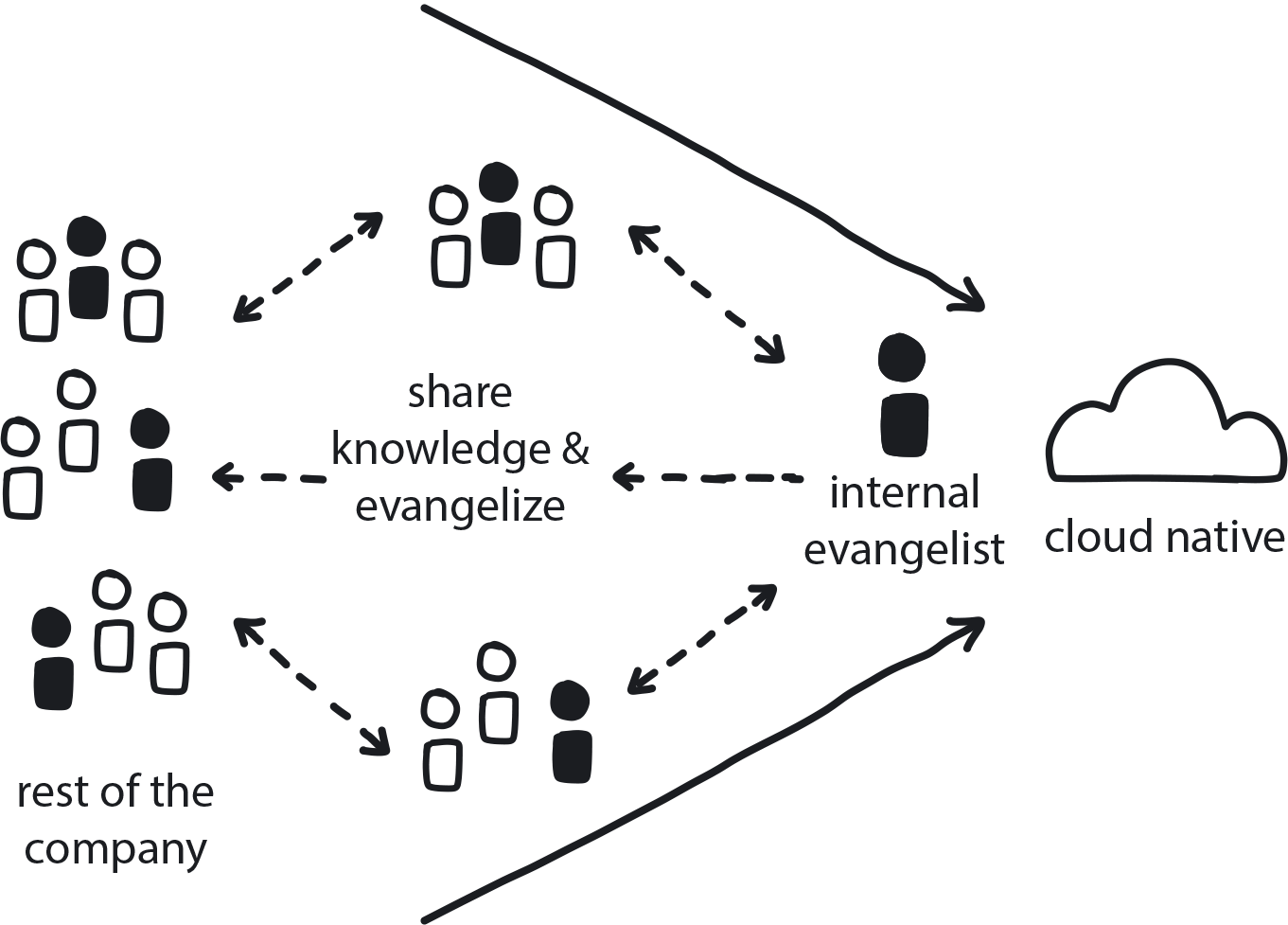
“This brand new tool/technique must be better than these boring old ones we have been using!”

**Related Patterns**

* [Manage for Proficiency](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#manage_for_proficiency)
* [Measure What Matters](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#measure_what_matters)
* [Three Horizons](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#three_horizons_pattern)

**Pattern: Internal Evangelism**

*Provide plenty of information about the transformation across the entire company right from the start to create understanding, acceptance of, and support for the initiative (*[*Figure 8-15*](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#internal_evangelism01)*).*



**Figure 8-15. Internal Evangelism**

Cloud native transformation is underway, and the Core Team is working hard to understand the challenge and build a high-quality platform with a clear way to onboard old and new applications to it. The rest of the teams are not yet involved in the transformation, and it may take a few months, even a year, before they are all onboarded to the cloud native new platform.

**In This Context**

**When there is little information about an ongoing cloud native transformation, people don’t automatically assume it’s a good idea. People don’t resist the transformation because they think it is the wrong thing to do—they resist because it is new and scary. Change creates anxiety, and most people just don’t know much about cloud native in general. Without clarity, people tend to fill in the gaps with imaginary negative information (negative attribution bias), and they may fear their jobs will be dramatically different—or even eliminated.**

Most people enter the transformation process from a traditional Waterfall hierarchy. In such organizations, managers have always just told them what to do. Cloud native teams, however, are highly independent and so require full participation and active involvement from all members. Without motivation to learn and do it right, the result could be disastrous for the company, as only a few teams will fully embrace the cloud native way of working.

* People have a lot of ways to resist by dragging their feet or even actively sabotaging an initiative.
* People need to be encouraged to behave differently.
* Punishment works to stop behaviors, but it will not work to inspire and engage people with new behaviors.
* People do not automatically receive a new message. They need many touch points to hear the new story until they accept and internalize it.
* Negative attribution bias: if you aren’t telling them about the change, they assume no progress is being made/nothing is happening.

**Therefore**

**Share positive, clear, and abundant information about the transformation to create acceptance, support, and even excitement across the company.**

Metamorphosing an organization’s tech and especially culture is a tribal movement. If people are not sure whether it’s a good idea, then you need to help them understand. Negative attribution bias can be powerful here: without enough info, you fill in the gaps with negativity. The solution is to fill in the gaps with a lot of information all the time—not just about what is happening, but why, how these changes will benefit the entire organization.

* Organize events, send newsletters, show demos. Essentially, do internal marketing.
* Tell the story over and over in a positive and inspirational way, rather than a “do this or else” way.
* Internal Evangelism should be led by someone who is committed and knowledgeable and involved. This person makes it public, involves people, and creates the movement.
* It’s imperative the evangelist demonstrates good knowledge of the business itself and what it does now, so their opinion is trusted as being informed and based on thoughtful judgment.
* The Transformation Champion is a possible candidate for this role, but only if it does not distract from their duties leading the transformation itself.

**Consequently**

**The transformation is understood across the organization, and people feel motivated to join and support it. There is plenty of time and opportunity to mitigate any resistance originating through fear of uncertainty.**

* + People have had time to get comfortable with the idea.
* + The message starts out very simple and gradually grows more detailed as both the project and the acceptance progress.
* + Small projects and experiments are opportunities for evangelism.

**Common Pitfalls**

No one actually promotes the initiative across the whole organization. Engineers may be hacking around building amazing systems but then they never talk about it. They don’t think it’s their job to tell others, but it is—part of the job of building anything is to make it successful. Not just rolling it out one day, but telling people what you are doing and why. The best way to see cloud native transformation is to consider it an internal startup that not only needs to build the platform but also to sell to internal customers.

The management version of this is to buy a very expensive, comprehensive tool and just force it on everyone, including the team that just built a very nice, small-scale custom solution that is perfect for the company’s needs. This is actually one of the biggest problems in transformations and easily avoidable with Internal Evangelism—yet very few companies do it.

**Related Biases**

*Curse of knowledge*

Transformation information has to be understandable to the people you are seeking to sway. You already know you need to simplify the message when broadcasting cloud native transformation info across the organization. But because you are deeply involved in the initiative, with expert knowledge level of 9 on a scale of 10, you boil the message down to 5 or 6 when talking to others, even if they only understand it to maybe 2.

*Negative attribution bias*

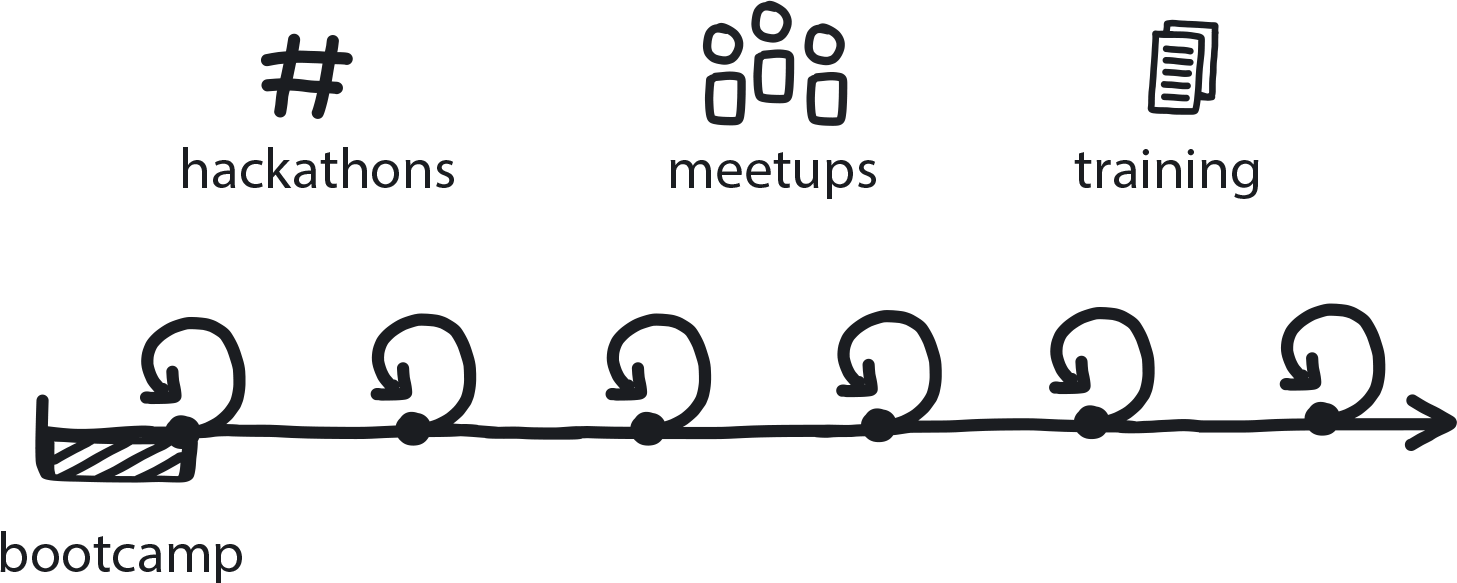
People assume that if they are hearing nothing about the transformation, then no progress is being made/nothing is happening.

**Related Patterns**

* [Transformation Champion](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#transformation_champion)
* [Executive Commitment](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#executive_commitment)

**Pattern: Ongoing Education**

Continuously introduce new ways and improve existing ones to help teams continually develop their cloud native knowledge and skills ([Figure 8-16](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#ongoing_education01)).



**Figure 8-16. Ongoing Education**

The company is moving to cloud native, and some teams have never worked with cloud native technology or processes. In their previous environment, knowledge and learning were reasonably stable and linear (learn a bit and exploit the knowledge a lot).

Other teams are already deep into building the cloud native platform or microservices and so have gained basic cloud native knowledge, but still are not advanced enough to support the entire transformation.

**In This Context**

**People are joining the organization’s cloud native initiative without fully understanding the possibilities it offers or the wide variety of solutions available. New technology is introduced all the time that renders current tools and techniques out of date. When this happens, productivity suffers, and change slows down.**

* Only some people are motivated to learn on their own.
* People learn better in groups.
* Formal education is more effective after some initial exposure to/experience with new information.
* Most people need to learn something a few times before they fully understand it.
* The cloud native ecosystem is changing fast.

**Therefore**

**Build and continuously run an education program about cloud native for everyone in the company, from basic education for newly onboarded or new joiners to continuous and more advanced trainings for more experienced engineers.**

* Onboarding bootcamps
* Hackathons
* Periodic knowledge updates
* Management trainings
* Online self-learning opportunities
* Books, blog posts, and other reading
* On-the-job learning by pair programming, whiteboarding, etc.

**Consequently**

**Team knowledge is constantly refreshed and updated.**

* + Easy ways to roll out new information.
* + A lot of inspiration arises for trying out changes.
* + Developers are better able to adapt to technology shifts.
* + Technology changes, which are inevitable, will be minimally disruptive since developers are better able to adapt to shifts in tech.
* + By disseminating best practices and successful ways of using technology, you can replicate success and avoid missteps.
* − There is cost related to frequent education.

**Common Pitfalls**

In most companies, teams onboarded to the cloud native platform would typically get at least some training, even if it is very simple and incomplete. The more common problem is with the engineers who are perceived to be cloud native experts after spending 6 to 12 months building the initial cloud native platform. While they would know significantly more about the cloud native field than their peers, they still need to learn much more before they become real experts. Engineers working in the cloud native field, and any other innovative areas, should continue their education indefinitely.

**Related Biases**

*Authority bias*

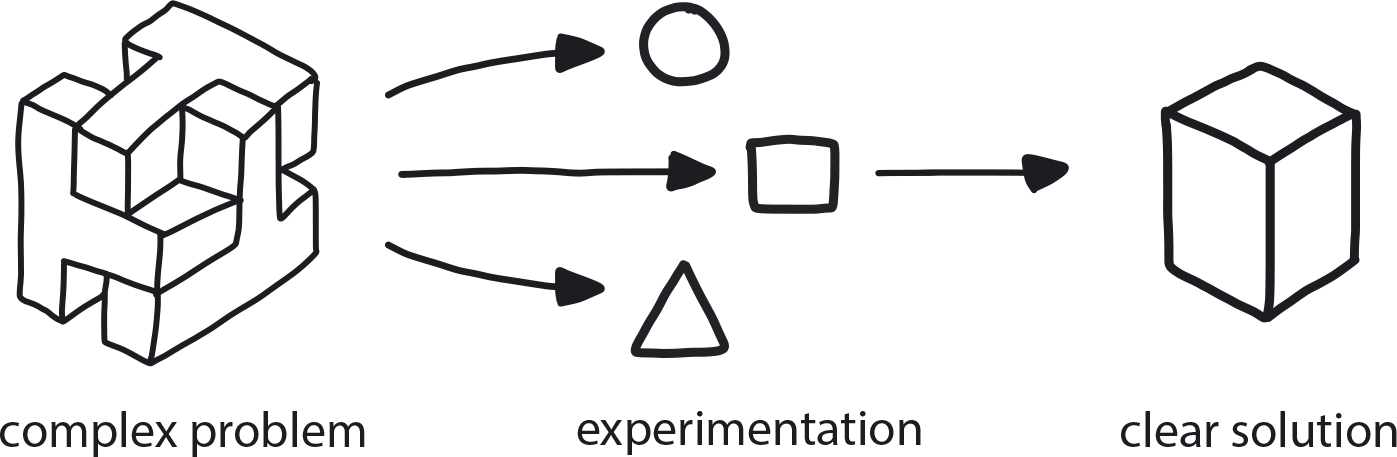
The tendency to attribute greater accuracy to the opinion of a perceived authority figure, whether or not they are truly expert, and thus be more influenced by that opinion. In this situation, the company may overly rely on the perceived expertise of the Core Team in making key decisions, when they might not yet have sufficient knowledge and experience to make informed choices.

**Related Patterns**

* [Learning Organization](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#learning_organization)
* [Internal Evangelism](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#internal_evangelism)
* [Demo Apps](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch09.html#demo_applications)
* [Gradual Onboarding](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#gradual_onboarding)

**Pattern: 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 (*[*Figure 8-17*](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#exploratory_experiments01)*).*



**Figure 8-17. Exploratory Experiments**

The challenge is new and complex, and the team can’t solve it by using its existing knowledge or through simple research (i.e., you can’t Google it). There is not enough information to make the next step, much less a proper decision.

**In This Context**

**Committing too early to a solution you don’t yet fully understand. Teams are likely to do one of three things: Choose a known solution that is not a good fit for the problem, because they are familiar with this solution; undertake a lengthy analysis of the solution that leads nowhere (analysis paralysis); or else jump on the first available solution (availability bias).**

* In Waterfall all questions need to be answered before actions are taken.
* People learn best by doing.
* Difficult problems are better solved by a team.

**Therefore**

**Explore the problem space. Mitigate the risk by delaying critical decisions long enough to run a series of small scientific-style experiments to uncover missing information and evaluate alternatives.**

Each experiment should have a hypothesis that can be proved or disproved by the experiment. When a hypothesis is disproved, the experiment itself can still be called a success. It is critical to avoid assigning blame or punishing those involved in the experiment in cases where the results are not satisfactory.

* Identify one problem/idea and choose two to five small—taking a few hours to a few days—experiments to test it.
* Form a hypothesis to query: *What is my problem in the first place, does this solve it, what is the cost, and can we afford it?*
* Identify and compare possible alternative solutions.
* Establish clear and measurable criteria for success or failure. Collect data and evaluate it.

**Consequently**

**The team is granted time and given a process for experimenting with solutions when it encounters a complex problem.**

* + Teams learn by doing this, which helps them in future decisions and work.
* + By showing a small, quick success with an experiment, you are able to make a first small and easy step in a new direction.
* − When experiments are cheap and easy enough to run, it can be hard to determine when it is time to stop experimenting and make a decision/move forward.

**Common Pitfalls**

If the problem is known or the solution is simple, running experiments is a waste of time and a diversion from actual work. Avoid experimentation simply for the sake of experimenting.

Can lead to endless experimentation and analysis paralysis when no decisions are made, as there are always plenty of assumptions to test and tools to evaluate.

Focusing on tactical experiments (which server to provision, which tool to choose) instead of tackling the big, difficult problem.

When there are multiple options but only one is realistic or available, experimentation is a waste of time if the path is already determined.

Introducing experimentation in teams that will not be moving to the new system soon can create unrest.

**Related Biases**

*Confirmation bias*

The risk of running experiments to prove a point instead of checking alternatives. If you are testing only one hypothesis, you’re not experimenting. You are jumping to a solution because it’s easier, which is S1 thinking. This book’s entire purpose is to point you to S2 (see [Chapter 4](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch04.html#beyond_patterns_behavior_biases): Behavior and Biases).

*Shared information bias*

Everyone talks only about the things that are already commonly known and then chooses a solution without considering and experimenting with lesser-known alternatives.

*Information bias*

The tendency to seek information even when it cannot affect action. You need to limit the experiments and make decisions at the end of the experiment to avoid analysis paralysis.

*IKEA effect*

If you get teams involved in building an experiment, there will be a bigger commitment to the idea, for two reasons. First, they may be reluctant to abandon an experimental version even when it is complete, because they built it themselves. Second, involvement in building the early portions of a new cloud native system can generate greater ownership of and support for the entire transition initiative.

*Irrational escalation/Sunk-cost effect*

Keeping experiments small before making a decision makes it easier to drop the results of any given experiment because there has only been a small investment.

*Status quo.*

Can use this as a nudge: by showing a small, quick success you can take a first small and easy step away from the status quo.

*Parkinson’s law of triviality, a.k.a. “Bikeshedding”*

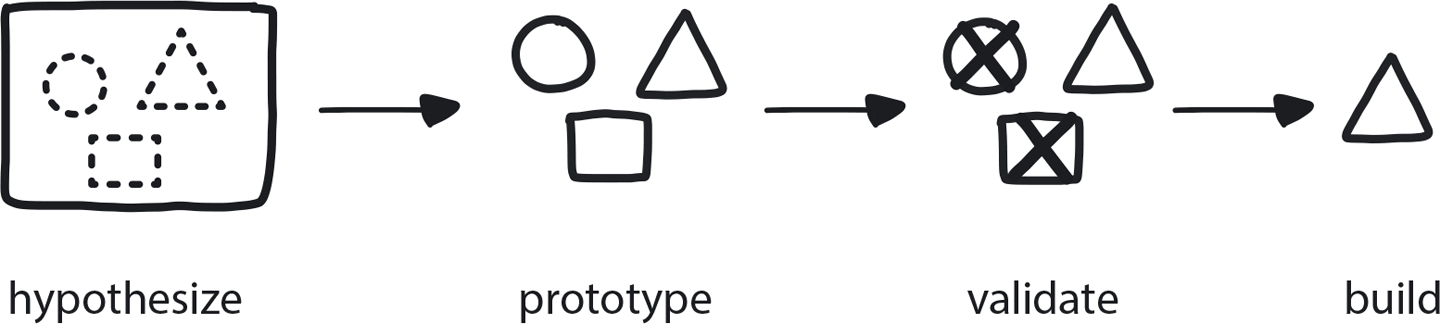
Choosing something easy but unimportant to test and evaluate, over something else that is complex and difficult but meaningful.

**Related Patterns**

* [Proof of Concept (PoC)](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#proof_of_concept)
* [Reduce Cost of Experimentation](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#reduce_cost_of_experimentation)
* [Design Thinking for Radical Innovation](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#design_thinking_for_radical)
* [Learning Organization](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#learning_organization)
* [Blameless Inquiry](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#blameless_inquiry)
* [Manage for Creativity](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#manage_for_creativity)

**Pattern: Proof of Concept (PoC)**

*Before fully committing to a solution that can significantly affect the future, build a small prototype to demonstrate viability and gain a better understanding (*[*Figure 8-18*](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#proof_of_concept01)*).*



**Figure 8-18. Proof of Concept**

You have run experiments and identified a solution path that you think could be the right one, but there are still some big unknowns. You are at a decision point: adopt this solution, or not?

**In This Context**

**Once some initial experiments have uncovered a likely transformation path, it is time to test it. Committing to it right now could, if it is wrong, cause large problems and expense as the initiative continues.**

**You simply don’t know enough to make a large commitment at this point. Any full commitment right now carries massive risk because switching to an alternative later will be very difficult. Adopting a solution you don’t fully understand too early in the process compounds the risk, because you will continue to build further functionality on top of this solution.**

* Hands-on work, rather than promises or explanations, is better for demonstrating value to skeptics.
* Changing early decisions is costly in the later stages of a migration project.

**Therefore**

**Build a basic functional prototype to demonstrate the viability of a solution. Define the questions that need answers before starting the PoC and stop the work once the questions are answered.**

* This should take a few days to a few weeks.
* Build the most primitive version you can get away with. Be ready to throw it away when you are finished. (This doesn’t that mean you *must* throw it away, but often when initial quality is intentionally low, it’s cheaper and easier to just scrap it and rebuild from scratch).
* Only work on hard issues related to the specific needs of the current project, and stop when the solution is clear.
* Try to demo something functional so you can collect business-related information regarding how this solution will impact future development.

**Consequently**

**Risk is reduced for the overall project in the early stages of the migration. Critical knowledge and experience are gained during the experimentation process.**

* + You have gained the knowledge and proved the solution works, and now understand how it fits into the overall project.
* + You now have reasonable confidence that this is the correct decision.
* − Running PoCs carries cost. Every time you do it, you pay for it.

**Common Pitfalls**

Can be used to enforce existing beliefs (confirmation bias). This is how most PoCs are run, unfortunately—only one option is tested.

A team spends three days proving a concept and has the answer, but keeps going to reach a fully functional version. This is a waste of time. Once you understand it and the solution has proved viable, pull the plug and move on. Doing this is especially difficult for engineers.

You take the prototype to production, even though it was intended as a basic experiment.

**Related Biases**

*Sunk-cost fallacy*

Once you’ve put this much effort into something, you just want to keep going even though you have your answer already.

*Confirmation bias*

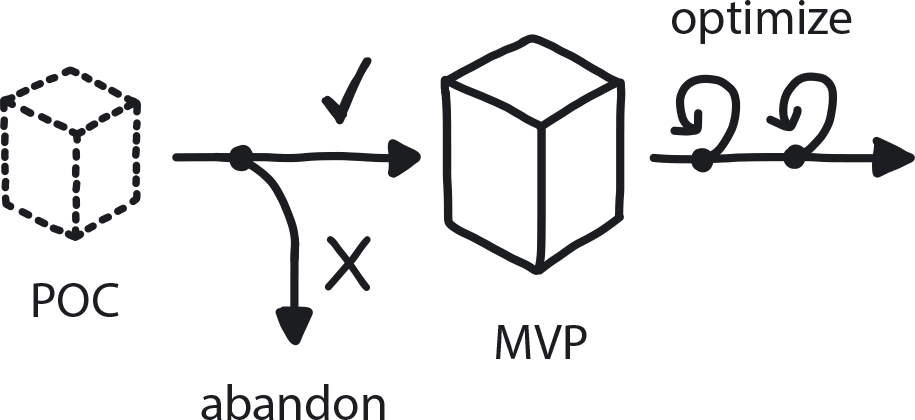
You think you already know the right answer so you embrace the information that supports your opinion (while ignoring any inconvenient facts that might disprove it).

**Related Patterns**

* [Exploratory Experiments](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#exploratory_experiments)
* [MVP](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#mvp_platform)
* [Gradually Raising the Stakes](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#gradually_raising_the_stakes)
* [No Regret Moves](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#no_regret_moves)
* [Options and Hedges](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#options_and_hedges)
* [Big Bet](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#big_bet)
* [Objective Setting](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#objective_setting)

**Pattern: MVP Platform**

*Once Exploratory Experiments and PoCs have uncovered a probable path to success, build a simple version of a basic but fully functional and production-ready platform with one to three small applications running on it in production (*[*Figure 8-19*](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#mvp_platform01)*).*



**Figure 8-19. MVP Platform**

The goal is clear, the tech is available, and knowledge is present, and all major questions are answered by a series of PoCs. Now is the time to build a real platform. Development teams are on hold and waiting for the new platform to start building new applications and refactoring old ones.

**In This Context**

**Trying to add too many functions to the first release will make it very protracted and delay any release to production.**

**In the traditional Waterfall approach a new platform will be used in production only if it’s fully finished and everything is ready. Building a fully featured cloud native platform may take up to three years, but since cloud native components work independently, the platform can be running apps in production even before it’s completely built out. Not using the platform or any of its features until it is 100% complete would be a lost opportunity.**

* Minimum basic functionality could be built in a fraction of the time it takes to build the full system.
* There is always something to remove from the list of critical features.
* Custom complex solutions are difficult to build and maintain.

**Therefore**

**Define and build a system with minimal useful—and production-ready—functionality on a quick timeline. It’s important to release something quickly in order to start getting real feedback so you can then improve things based on that user response.**

* “Minimal usefulness” can vary per company.
* Cloud native Platform MVP should take two to six months to produce.
* Build good basics and quality to reduce the need for user support, but don’t go overboard.
* Extendability is critical.
* Plan additional two to three phases to bring the platform to full functionality.
* Use experiments and PoCs to define the MVP.
* Stress assumptions in real life**—**build a system that works with how people will need and want to use it.

**Consequently**

**The first MVP of the platform is ready and can be used by a small number of development teams to deliver a few apps to production. The Platform Team has a plan to expand the scale and functionality of the platform as it continues rolling it out to the rest of the organization.**

* + Teams can begin learning the basics of using a cloud native platform even while the final production version is still being developed.
* − The MVP represents 20%–40% of the final platform; further effort is still required to build out a complete, production-ready platform.
* − Core team needs to do support for this version while continuing to develop the production platform that will eventually replace it.

**Related Biases**

*Default effect*

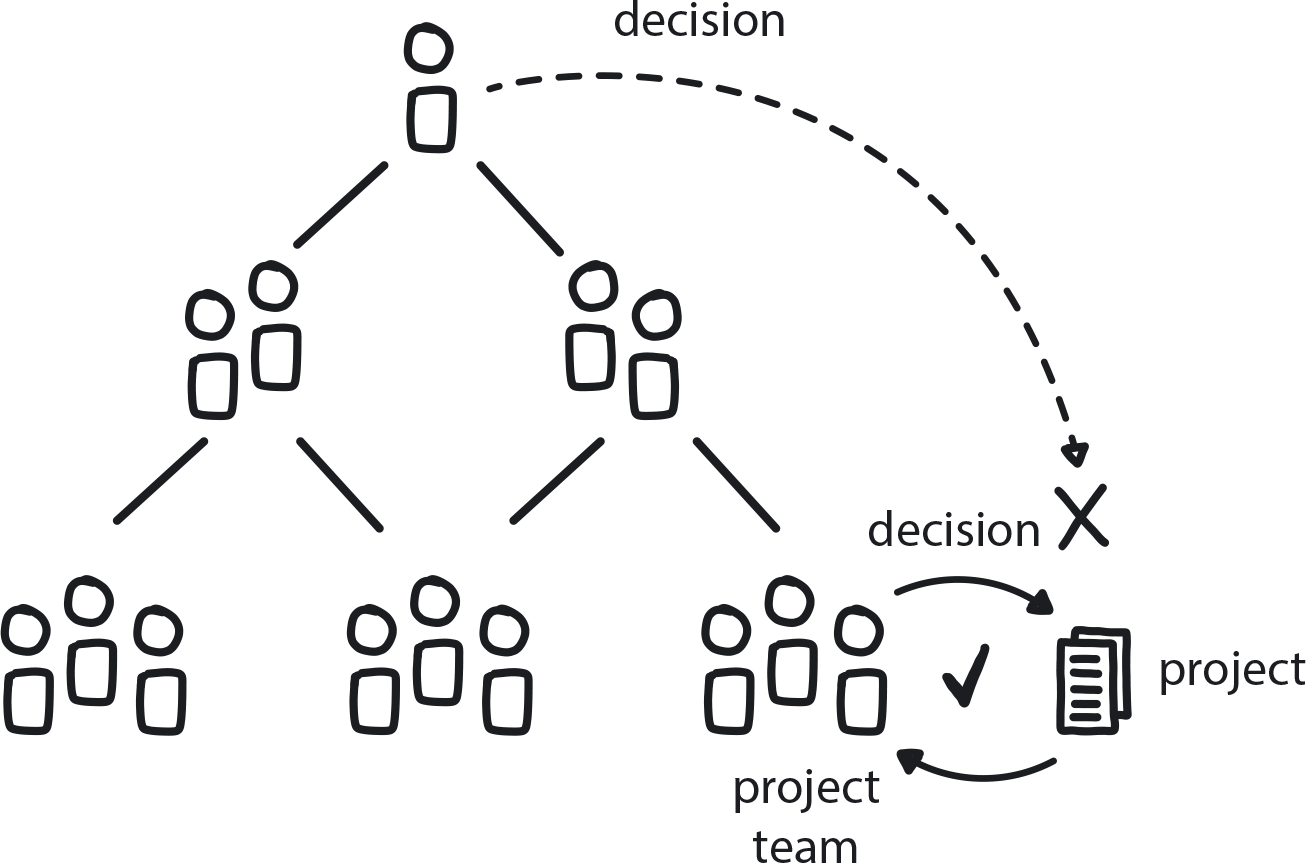
The Core Team will need to carefully configure the platform to be as optimal as possible before beginning to onboard the first teams. Users tend to simply accept the default options and settings provided to them as “the way to do things,” and only an adventurous few will try changing or customizing them.

**Related Patterns**

* [Exploratory Experiments](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#exploratory_experiments)
* [Proof of Concept (PoC)](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#proof_of_concept)

**Pattern: Decide Closest to the Action**

*Those nearest to a change action get the first chance to make any decisions related to it (*[*Figure 8-20*](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#decide_closest_to_the)*).*



**Figure 8-20. Decide Closest to the Action**

Cloud native transformation is underway, and there is a lot of uncertainty. People are still learning about the tech, which itself is continually evolving, and the market is changing frequently and erratically. Each team is responsible for delivering its own microservice, and there are many moving pieces. Managers and lead architects have only a broad, high-level understanding of the product and little grasp of the technical details that underlie the actual development process.

**In This Context**

**Decision making via chain of command is not sustainable in cloud native. Using hierarchy to resolve conflicts and agree on decisions takes too long, and solutions are limited to the capabilities of the managers making that level of decision. Engineers might find a superior solution that never gets implemented because it takes too much time and effort to navigate the bureaucracy to get permission. So they will give up and just move on with whatever they have.**

* Recent speed of technological change is growing exponentially.
* Market is now also changing frequently, with unexpected new competitors appearing.
* In traditional organizations, managers make all the decisions and give instructions to engineers.
* Further you go from the change, the slower the decision over time.

**Therefore**

**Push the decision power as close as possible to any change as it is happening. It is typically best if the dev team itself makes the decisions.**

Because your team works as part of a bigger tribe of teams, sometimes your decision must involve others. For example, if you want to change APIs, you must inform those who use them—and they may object to the change. So whatever happens inside of a microservice is fully the domain of its development team, but anything going in/out is also the domain of the teams that consume them.

* Put in place security and regulation to make sure people are allowed to make these decisions.
* Complete separation of duties: execs in charge of strategy, managers in charge of setting objectives, engineers free to execute their work as they see best.
* Instill that it’s OK to fail into the organization’s values.
* Use hierarchical management for conflict resolution.

**Consequently**

**Executives delegate the power to create the vision and objectives to middle management, and middle managers delegate power over technical decisions to the execution teams.**

* + Strong incentive to resolve conflicts first within team, then within tribe, only then go to management.
* − Time and effort are required to coordinate with any teams who are consuming your particular service to make sure it works all around.

**Common Pitfalls**

Punishment for failure is the biggest mistake in most organizations. In cloud native, decision making happens fast and frequently, and mistakes are inevitable. Penalizing people when they make mistakes dramatically reduces their willingness to try new things, and they will move more slowly to avoid making errors. They will ask for permission, which will slow down or kill all innovation. If someone keeps failing in the same way, then there is need to intervene—but the company culture needs to allow failure without consequences.

Managers, especially executives, are used to being in charge and dictating solutions. In cloud native, leaders need to learn that teams often experiment their way to solutions and the path is not always clear. From the outside this can look like a lack of progress. It takes patience to let people make these decisions on their own, but higher-ups often feel frustrated and jump in to “fix” things too soon—thus derailing the team’s progress and learning.

In this pitfall we find a CEO who is too impatient for results or middle managers who can’t understand the strategy. In either situation the CEO feels compelled to step over the middle managers and micromanage the team. This sows confusion in the teams and undermines the middle managers, while dragging the CEO into a tactical hell.

Sometimes the executive team is too slow coming up with the strategy. In response, the rest of the company will usually just settle into the inertia or create their own guiding organizational strategy, which in turn is often shortsighted and causes more problems than it solves.

**Related Biases**

*Law of the instrument*

An overreliance on familiar tools or methods, ignoring or undervaluing alternatives. “If all you have is a hammer, everything looks like a nail.”

**Related Patterns**

* [Communicate Through Tribes](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#communicate_through_tribes)
* [Dynamic Strategy](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#dynamic_strategy)
* [Blameless Inquiry](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#blameless_inquiry)
* [Research Through Action](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#research_through_action)
* [Value Hierarchy](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#value_hierarchy)

**Pattern: Productive Feedback**

*People are more engaged and creative when they feel comfortable receiving constructive information about their behavior and giving the same in return (*[*Figure 8-21*](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#productive_feedback01)*).*



**Figure 8-21. Productive Feedback**

You have a team whose main responsibility requires creative or innovative work.

**In This Context**

**People are often blind to their own biases and live in their own bubble without realizing it.**

**During a cloud native transformation, teams that have always worked in a proficient way now are tasked with innovation. They have no experience being creative, so they will keep using past solutions to attempt to solve new problems.**

Lacking external perspective you will be blind to your own confirmation bias and interpret results to fit your preconceived notions, leading the project to very poor results. At the same time, other people within the organization can clearly see the problems before they happen.

* Judgmental tendencies are hardwired.
* Most people don’t give positive feedback unless prompted to do so.
* Without empathy, feedback can be perceived as aggression or attack.
* For people who have never given feedback it can be difficult to start.
* An existing team has already solidified their relationships and will find it difficult to change their behavior.
* Significant feedback requires some level of personal relationship.

**Therefore**

**Create a safe environment and simplify ways for people to give feedback—positive, negative, even confrontational—in a constructive way.**

Facilitate team activities to build personal connections between team members; this helps to create a sense of mutual trust that allows feedback to flow freely.

* Offer training or tools to teach people how to give constructive feedback.
* Create opportunities, such as one-on-one or group meetings or weekly email updates on the project, to exercise giving constructive feedback.
* If someone is doing something well, feel comfortable giving them praise.
* If you think someone is taking a wrong approach, tell them in a constructive way.
* Someone needs to act as leader to model how to do all this.
* Psychological safety is an essential prerequisite for Productive Feedback.

**Consequently**

**Productivity goes up because people can learn and improve their work and behavior, and because they feel seen and appreciated.**

* − There are costs involved: honest and constructive feedback relies on personal relationships and psychological safety.
* − High risk of personal conflict, but this is the chance you have to take if you really want to go fast. People will learn on their own without feedback, but it will take a lot longer, and some things they will never learn on their own.

**Common Pitfalls**

In a proficient delivery system Productive Feedback has little value because the process is algorithmic and innovation is not desired. Over time, the unspoken rule becomes “just shut up and do your own thing,” and this becomes part of the culture in traditional companies. In the creative environment required for a cloud native transformation, however, Productive Feedback is essential. People don’t really know how they are doing, and without feedback they can keep going in the wrong direction simply because it’s their best guess and they have no other information. We see this all the time in big organizations building a new platform that works well for the team building it but does not fit what other stakeholders need. Since they never expose the work in progress to feedback, however, this problem emerges only at the very end.

**Related Biases**

*Blind spot bias*

People can spot others’ biases, but not their own.

*Confirmation bias*

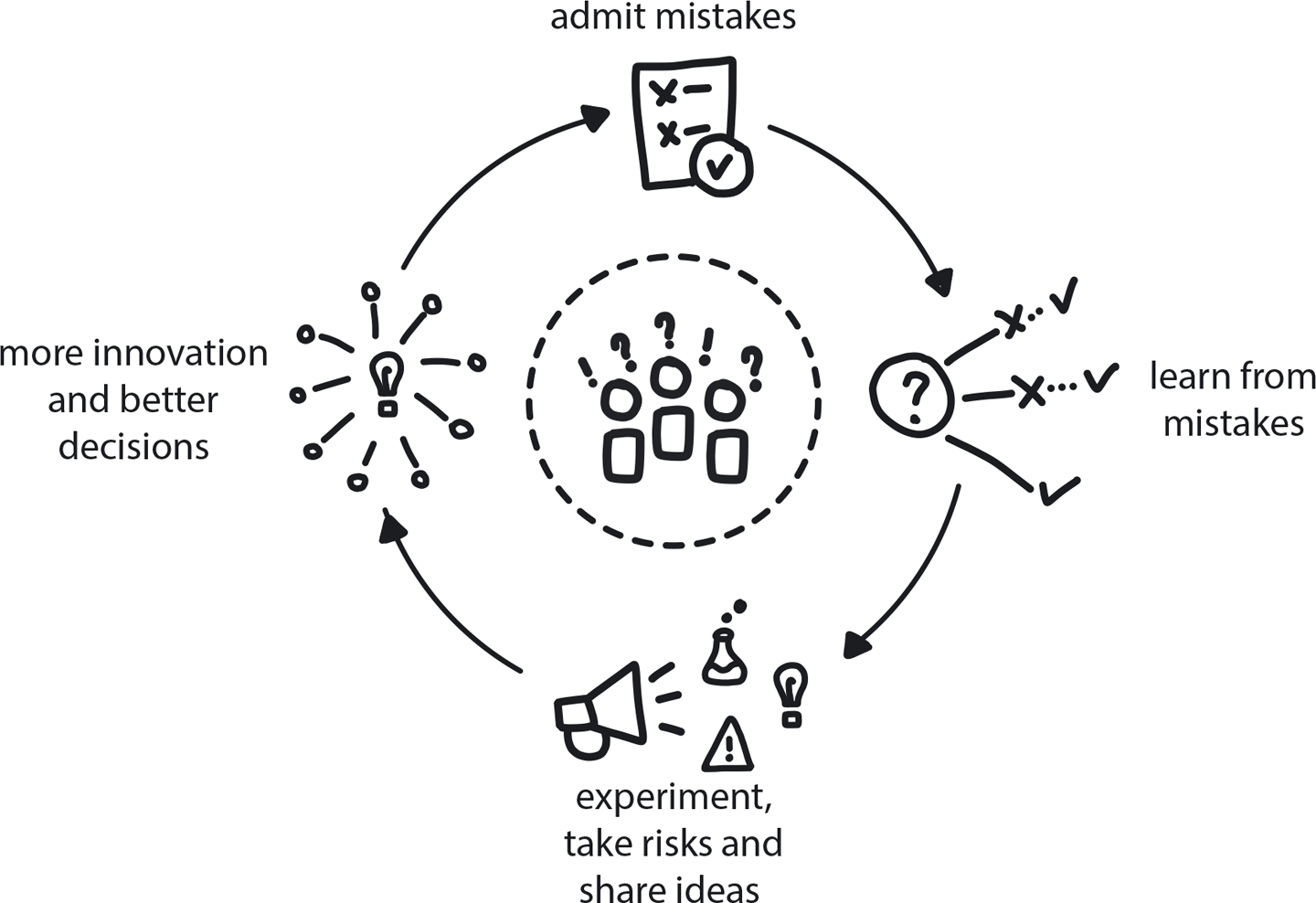
Seeking to “prove” your existing opinion, rather than genuinely exploring alternatives.

**Related Patterns**

* [Psychological Safety](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#psychological_safety)
* [Personalized Relationships for Co-Creativity](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#personalized_relationships_for_co_creat)
* [Learning Organization](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#learning_organization)
* [Blameless Inquiry](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#blameless_inquiry)

**Pattern: Psychological Safety**

*When team members feel they can speak up, express concern, and make mistakes without facing punishment or ridicule, they can think freely and creatively, and are open to taking risks (*[*Figure 8-22*](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#psychological_safety01)*).*



**Figure 8-22. Psychological Safety**

A company is about to embark on a complex journey toward cloud native. The path is uncertain and strewn with problems that the team has never encountered. The team needs to take a collaborative problem-solving approach, learn together, and push one another to think creatively.

**In This Context**

**In traditional enterprises that are mostly designed to support stability, people fear exposing themselves by asking a “stupid” question, suggesting a “crazy” idea, or giving difficult feedback to a teammate, let alone a manager. All such actions typically are dismissed and ridiculed or, even worse, punished.**

In such an environment people tend to keep their ideas to themselves until they’re fairly certain they will be welcomed by others. That means the best and newest ideas might never have a chance of being adopted. And since many new ideas seem a bit crazy and risky in the beginning, the team never gets a chance to really innovate.

* Experimentation requires honest assessment.
* Fear of failure generates resistance to trying new tools or techniques.
* Many attempts to discover new things will fail.
* In hierarchical organizations, people at all levels tend to hide information that could be harmful for their own careers.
* If teams lack the ability to fail safely when experimenting, they must waste time analyzing all future outcomes to minimize exposing themselves to risk.

**Therefore**

**Create the shared value that no group member will ever be punished or humiliated for speaking up with ideas, questions, concerns, or mistakes.**

In the workplace, psychological safety among teams means that the members believe they can express ideas, concerns, or mistakes without being punished or humiliated for it. As a result, they are willing to express new or different ideas, even if they diverge from overall group opinion. When we feel safe, we become more open-minded, resilient, motivated, and persistent. A sense of humor is encouraged, as is solution-finding and divergent thinking—the cognitive process underlying creativity. This creates the confidence necessary for trying and failing and then trying again. It also shuts down time-wasting risk-avoidance behaviors like trying to find a solution that is absolutely going to work before you even start trying it out. In cloud native that solution does not even exist, anyway.

* Many discoveries look odd and “stupid” at first glance.
* In an environment with high trust, people will feel confident in speaking up.
* Creativity occurs much more often in high-trust environments.

**Consequently**

**People can propose new methods or approaches knowing that their ideas will be treated respectfully.**

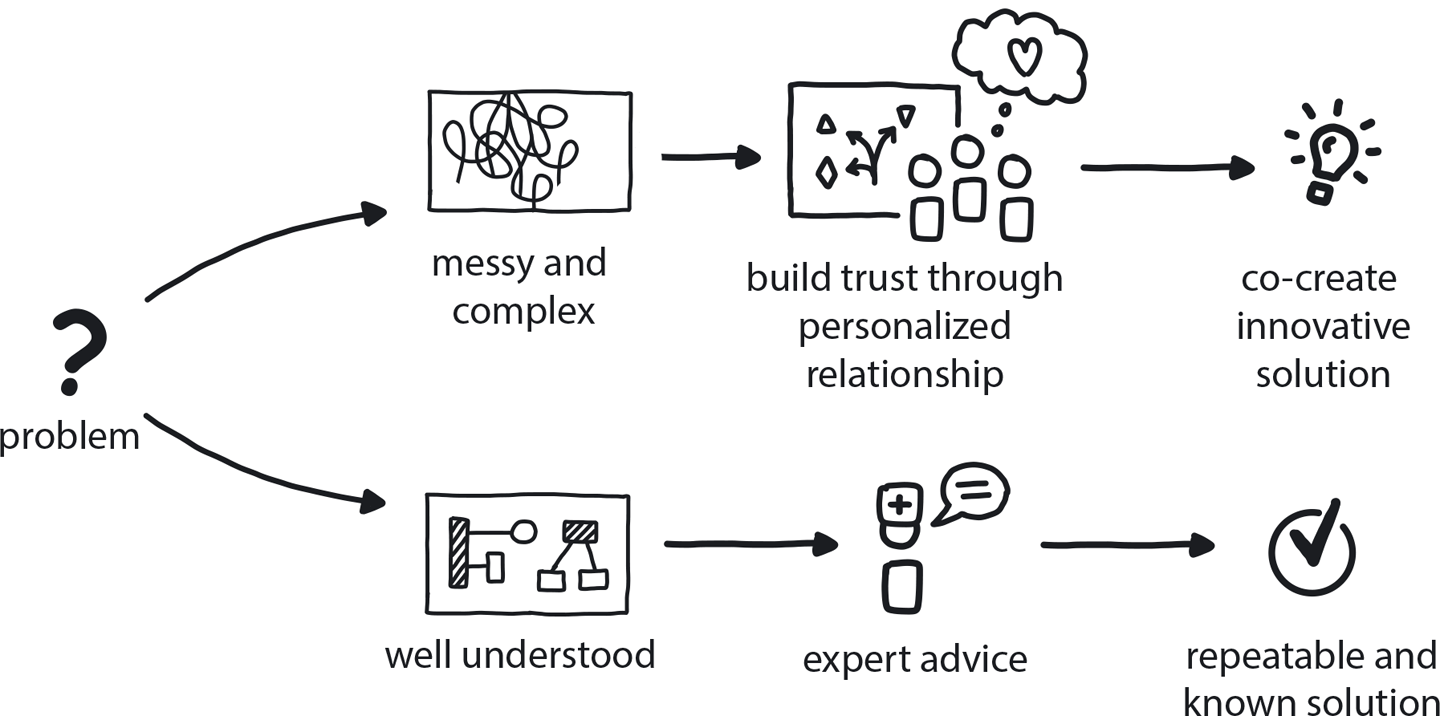
* + Team members can put forward “wild and crazy” ideas without fear of shaming or embarrassment.
* + Even when experiments fail, learning from them via Blameless Inquiry feeds further improvements.
* − Creating psychological safety in an organization requires intentional investment of time and effort.

**Related Patterns**

* [Productive Feedback](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#productive_feedback)
* [Blameless Inquiry](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#blameless_inquiry)
* [Learning Organization](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#learning_organization)
* [Manage for Creativity](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#manage_for_creativity)
* [Personalized Relationships for Co-Creation](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#personalized_relationships_for_co_creat)

**Pattern: Personalized Relationships for Co-Creation**

*Solutions to complex problems are best created collaboratively by teams with high levels of interpersonal connection (*[*Figure 8-23*](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#personalized_relationships_for)*).*



**Figure 8-23. Personalized Relationships for Co-Creation**

The team is working on the more creative part of the implementation and still is only partially knowledgeable about it. There are no immediate answers, and the future is not clear or predictable. Teams are based on personal expertise and have clear roles and responsibilities.

**In This Context**

**In uncertain environments, what worked in the past may not work here, so you need to invent rather than attempt to reuse existing solutions.**

* Creativity requires trust and volunteering information.
* By default people will use existing knowledge to solve a problem.
* People will not volunteer information to an expert.
* They will only volunteer when there is trust and a personalized connection.
* New solutions are best achieved in highly collaborative, small creative teams.

**Therefore**

**In complex environments where there is no clear path forward, a strong team needs personalized relationships to collaborate on creative solutions. Creativity is not the goal—co-creation is the goal. Creativity is open-ended and may not lead to anything, but co-creation generates results.**

Cloud native is better suited to co-creative relationships where the team itself collaborates on the future solution with all team members participating. People are solving the problem together rather than relying on the expert advice of individual members.

Co-creation teams are significantly more innovative and creative because people are thinking together and drawing from group understanding and experience. They work by building mutual trust, volunteering extra information, and being willing to take risks as a group.

* To create closer relationships, literally reduce the physical distance between team members. Teams should share the same workspace whenever possible.
* Run trust-building experiences, bringing people together to build personal relationships.
* Encourage collaboration and co-creation by working in groups.

**Consequently**

**The team has established trust and a relationship that helps people share information effectively, which leads to co-creating new solutions.**

* + Each person on the team is helping to create solutions.
* + The team as a whole is more creative than each individual.
* + People have individual expertise, but they contribute it to the wider team objectives.
* − Effort is required to bring people together for social events, extra activities, etc. Not just money and time, but also psychological safety and other organizational culture aspects that take investment.

**Common Pitfalls**

Assuming that just because people sit together, they are a team and will be effective working together. Thus not investing in any team-building programs or efforts, and then still expecting people to innovate.

Looking for a guru: When seeking solutions and figuring out requirements, a team can understandably turn to an individual with deeper knowledge or experience—an “expert relationship” delivering knowledge from above. Relying on the existing knowledge of individuals, even very knowledgeable ones, significantly hinders innovation and creative thinking. Once the knowledge is stable and well-understood, however, the expert relationship is useful in an economy of scale approach—it can be packaged in terms and names and executive strategy and rolled out to everyone.

**Related Biases**

*Bystander effect*

The tendency to think that others will act in an emergency situation. This is very relevant for overlapping responsibilities. In a Waterfall organization, when a task doesn’t officially belong to anyone, no one will volunteer to pick it up.

*Law of the instrument*

An overreliance on familiar tools or methods, ignoring or undervaluing alternatives. “If all you have is a hammer, everything looks like a nail.”

*Authority bias*

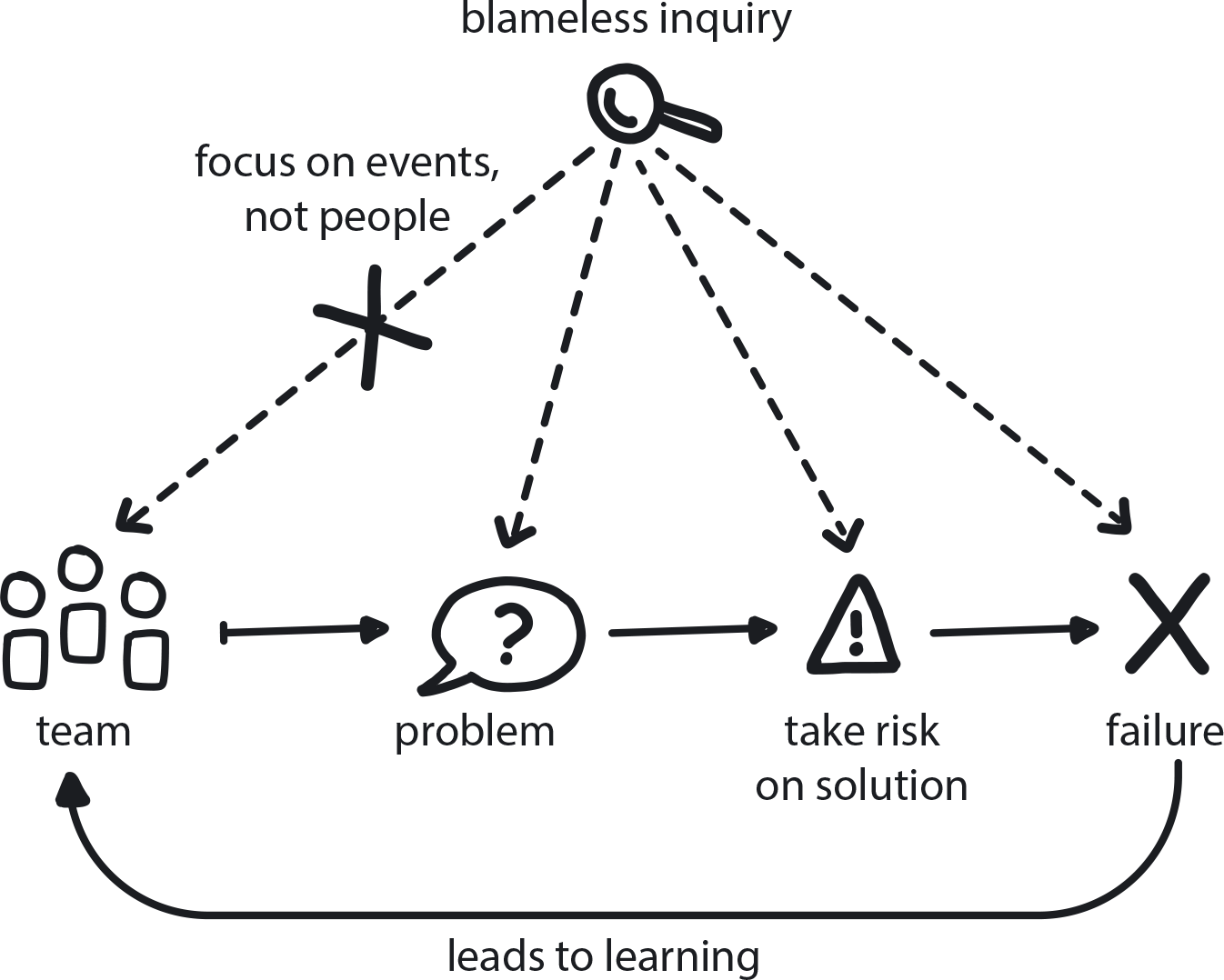
The tendency to attribute greater accuracy to the opinion of an authority figure and be more influenced by that opinion. In this context, seeking an “expert relationship” is falling back on the desire for someone with more experience to tell you what they think the right solution would be.

**Related Patterns**

* [Remote Teams](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#remote_teams)
* [Co-Located Teams](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#co_located_teams)
* [Communicate Through Tribes](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#communicate_through_tribes)

**Pattern: Blameless Inquiry**

*When a problem occurs, focusing on the event instead of the people involved allows them to learn from mistakes without fear of punishment (*[*Figure 8-24*](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#blameless_inquiry01)*).*



**Figure 8-24. Blameless Inquiry**

A company is proficiently delivering existing products or services while investing some resources into continued innovation. There is a lot of uncertainty, and experimentation is required for figuring out the tech and exploring new business opportunities. Naturally, many of the experiments lead to dead ends.

**In This Context**

**When no inquiry is done after a problem occurs or an experiment fails, the team doesn’t improve and is likely to keep making similar mistakes. In many organizations, fault-seeking occurs, and blame gets assigned to anyone involved with a problem. This leads to mediocre performance, since most innovative actions carry significant risk.**

* Once punished for failure, people tend to avoid risks.
* Waterfall organizations tend to punish teams or individuals associated with problematic events to ensure stability; novelty is discouraged.
* There is no innovation without risk.

**Therefore**

**Understand what went wrong by focusing on the problem instead of the people involved.**

Talk about what went wrong and why, and how to avoid it in the future. Don’t punish people for mistakes—encourage them to learn from the experience instead. This creates the psychological safety required for taking risks.

* Gather everyone involved and review what happened.
* Share the results with the rest of the team.
* Find possible solutions for avoiding similar problems in the future.
* Focus on the problem, not the people: don’t ask who did what and never assign blame.
* If some mistakes occur repeatedly, find out who is involved and introduce personal commitment/responsibility.

**Consequently**

**People have the autonomy and the confidence to try and fail, and try again.**

* + Teams are always improving by learning how to find and avoid repeated mistakes.
* + Progress happens faster since people don’t waste time and effort trying to find a solution that is “guaranteed to work” before they even risk trying it.
* − If there are no consequences for failure, some people may fail due to not taking personal responsibility for solving problems as they arise.

**Common Pitfalls**

The nature of experiments is that some prove the point in question and some disprove it. The biggest pitfall in this context is to blame and punish the people running the experiment for a negative result. This leads to fear of failure and significantly reduces innovation, because most people who have experienced punishment will subsequently run experiments only when they are certain of the outcome.

**Related Biases**

*Confirmation bias*

Picking test variables that will “prove” your existing opinion, rather than genuinely exploring alternatives.

*Congruence bias*

If you have a preconceived outcome in mind and test results that confirm this, you stop testing instead of seeking further information.

**Related Patterns**

* [Psychological Safety](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#psychological_safety)
* [Learning Organization](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch07.html#learning_organization)
* [Manage for Creativity](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#manage_for_creativity)
* [Manage for Proficiency](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch08.html#manage_for_proficiency)

**Summary**

In this chapter we introduced patterns around organizational culture and structure. The intent is to first expose readers to the patterns themselves before applying them—fitting them together in the transformation design outlined in [Chapter 11](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch11.html#applying_the_patterns) and [Chapter 12](https://learning.oreilly.com/library/view/cloud-native-transformation/9781492048893/ch12.html#applying_the_patterns_a_cloud_native). There, we show how a company like WealthGrid can apply patterns step by step, from start to finish, as a transformation design. The design moves the company through four stages to emerge successfully transformed into a flexible, responsive, and above all confident organization, able to work both proficiently and innovatively as needed.

When working in a design context, this chapter—along with the other chapters presenting patterns for strategy, development, and infrastructure—functions as a more in-depth resource for referencing and working with particular patterns.

New cloud native patterns are emerging constantly. To continue sharing them and extending the cloud native pattern language we have established here, please visit [www.CNpatterns.org](http://www.cnpatterns.org/).

This is where to find the latest pattern developments, but also an online community for discussing and creating new patterns. We are inviting people from across the industry, thought leaders and influencers but most importantly everyday engineers and managers—those out there working elbows-deep in cloud native code and architecture—to contribute and participate. Hope to see you there!