A holistic approach to DevOps

Our conquests, learnings, and experiments

Table of Content

Executive Summary

Today, embracing DevOps is more than applying certain practices and principles. Like everything else, it has evolved into a culture that combines management practices in lean and agile principles with technical practices in continuous delivery. Having inculcated a DevOps culture in our DNA, we at Nagarro bring you certain guiding structures and learnings that are a result of experiments using existing ideas and concepts. Let's dive deeper into these frameworks to discuss and ultimately improve the ways of working by creating better value during these fast-moving times.

01. Introduction: Embracing DevOps at the core

More than a decade ago, when DevOps was first on everyone's radar, a shift in attitude started to take place that essentially propelled the ad-hoc waterfall or a non-agile mindset to realign. Lean and agile ways of working were on the rise, but collaboration across organizations' hierarchical boundaries with diverse and (nowadays) non-combinable objectives was soon to be forgotten.

After a legendary appearance by John Allspaw and Paul Hammond at the Velocity conference in 2009, where they presented "10+ Deploys per Day: Dev and Ops Cooperation at Flickr", the first official DevOpsDays conference was held in Belgium, organized by Patrick Debois. He was an IT consultant with responsibilities in testing. And two years prior to this, he was caught between the worlds of development and operations and needed to find a solution that would allow him to complete a data center migration in the best possible way. It was a problem that could not be solved simply by moving chairs around.

Many evolutionary steps later over the past ten years, the realization has dawned that the silos were not just about development and operations. Information security, which until then had been mostly treated poorly, was approached, and brought on board alongside testing and quality assurance in general. It was only at the end of 2017 that Forrester proclaimed 2018 the year of enterprise DevOps. It was assumed that more than 50% of enterprises worldwide had already completed its DevOps transformation or were in the process of doing so.

Today, DevOps stands for organizational culture and business leadership that combines management practices in lean and agile principles with technical practices in continuous delivery. As the DevOps ecosystem evolved, we at Nagarro ingrained DevOps in our DNA quite early on. We remain committed to supporting our clients in their DevOps transformation and make it as seamless as possible. In fact, over the past couple of years, driven by our thinking breakthroughs values, we have been experimenting to see what new ideas and concepts can help us combine a fast-moving industry and the quality standards we uphold for ourselves.

02. Building an emergent and adaptive way of working

As DevOps evolved and the emphasis on management practices increased, we identified a few guiding principles or helpful mental models that are widely applicable. These have emerged over the years to help reason and understand how organizations can better deliver software products. While these principles may not be ready-to-use answers or blueprints, they do function as a helpful framework to discuss and ultimately improve the ways of working by creating better value, faster, safer, and happier.

These guiding principles are:

2.1 **Adopt a team-first approach** – Acknowledge that efficient delivery in complex environments can only be achieved with high-performing and stream-aligned teams.

2.2 **Identify Value Streams** – Consider orientation around value streams and business domains as a primary force when designing potential team structures.

2.3 **Accept cognitive capacity as a limiting factor** – Teams have a limited cognitive capacity that, in many situations, is the primary constraint for software delivery.

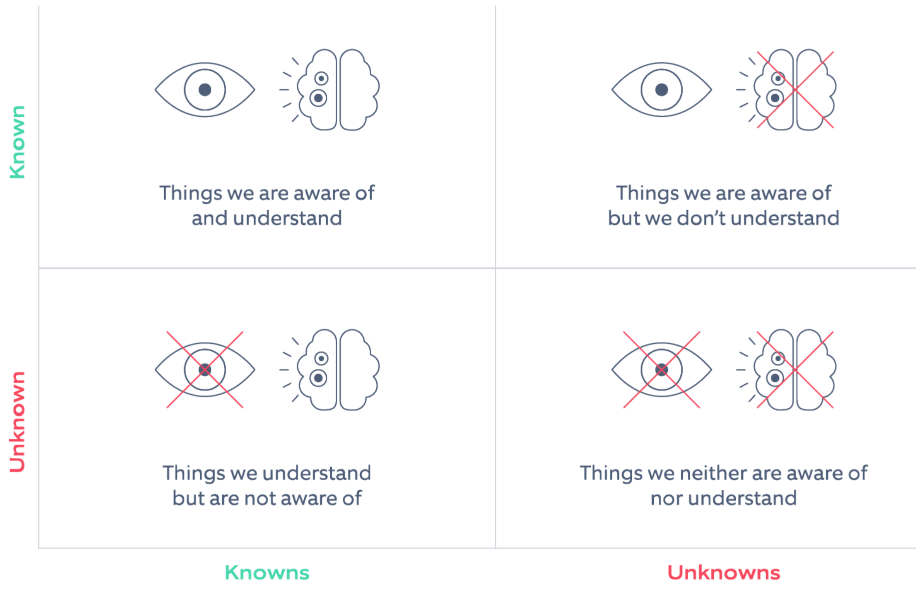
2.4 **Foster developer experience** – Focus on a good developer experience enables efficient delivery

2.5 **Enable teams** – Enable teams to work as autonomously and as self-reliantly as possible to reduce congestions and improve the overall flow, perhaps one of the most essential responsibilities of an organization.

2.1 Adopt a team-first approach

Among the most difficult and yet easy to understand areas of change in behavior and thinking is to establish small, long-lasting teams at the center of everything we do. This “team-first thinking” is also an essential aspect in “Team Topologies”, published by Matthew Skelton and Manuel Pais. Putting so much focus on teams also implies that we need to foster an environment to function correctly. However, we continue to see harmful practices and limiting organizational structures impeding teams from working effectively in many cases. For instance, some scenarios may include highly interdependent component teams, persistent knowledge silos, cognitive overload caused by too many different responsibilities of teams, and insufficient or unclear interfaces between teams or departments.

Many of our projects with our clients are not greenfield projects. In several cases, we deal with legacy systems, middle management with clear expectations, and team structures already in place. These client situations are complex domains that must deal with unknown-unknowns.


Known
Unknown
Things we are aware of and understand
Things we are aware of but we don't understand
Things we understand but are not aware of
Things we neither are aware of nor understand
Knowns
Unknowns

Orders of knowledge

Fig 1: Orders of Knowledge: Adapted from Donald Rumsfeld

There is emergence. There is no such thing as one-size-fits-all for organizational DevOps. There is no silver bullet. There is no one way, no DevOps-in-a-box that optimizes outcomes in all contexts. There is a need to focus on the benefit hypothesis and fast feedback to maintain flexibility and pivot to achieve the desired outcomes optimally.

Establish sustainable ways of working: Two concepts

Besides the general team-first thinking, we found two concepts beneficial to identify and communicate sustainable ways of working when designing or optimizing organizational structures:

- Cynefin Framework
- Conway's Law

Cynefin Framework

The Cynefin Framework, developed by Dave Snowden in 1999, is a conceptual framework used as a decision-making aid. It helps us understand the context in which we are operating. Cynefin provides five decision contexts or domains:

1. Simple
2. Complicated
3. Complex
4. Chaotic
5. And a center of disorder

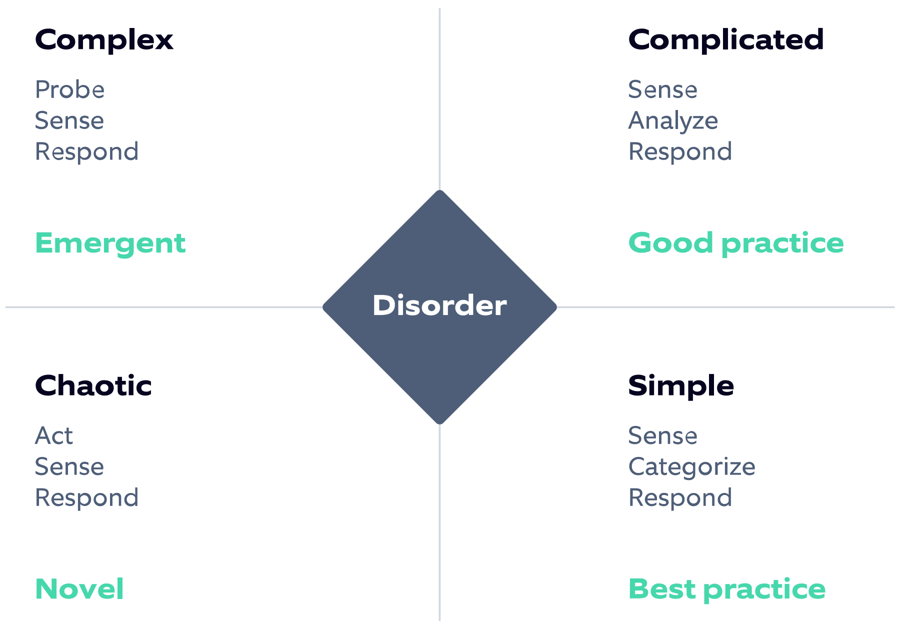


Fig. 2: Cynefin Framework

The domains provide a sense of place to analyze behavior and make decisions. The areas on the right, simple and complicated, are ordered. The cause and effect are known or can be discovered. The areas on the left, complex and chaotic, are disordered. The cause and effect can either be inferred retrospectively or not at all. This context, wherein decisions are made, is constantly changing.

The solutions to problems in the simple domain are obvious, and thus a correct approach is "sense-categorize-respond" by applying established and well-described best practices. There is no deeper analysis, experience, or expertise needed to categorize and respond. For various reasons, we sometimes persuade ourselves (or are persuaded) to falsely believe a problem is located here. But often, the more critical issues need a fundamentally different approach.

The complicated domain, where we are facing known-unknowns, is the ideal place for lean ways of working. For instance, the "right" or "correct" answers can be found in the Simple domain; however, additional expertise is required to identify the answer. Repetitive work is well-known and understood. It has been done many times before and requires expertise (e.g., installing a server in a data center). We know what to do when something goes wrong. It's deterministic, and the relation between cause and effect can be identified.

When dealing with sociotechnical systems, a direct relationship between the cause and effect can hardly be analyzed but only be deduced retrospectively. It is essential to state that this limitation is nothing that can be lifted by spending more time on analysis – it is inherent to problems in the complex domain.

Software in the digital age is not about writing the same code a thousand times. Often, the code is written once and executed a thousand times. Product development is unique; it has never been done before, at all or in context. This is the sweet spot where we use agile practices to keep evolving. In complex environments, there are no best practices and success is based on regularly analyzing the status quo and reflecting and adapting the way we work.

Conway's Law

An observation named after the US-American computer scientist Melvin Edward Conway, this law is based on the concept that the structures of the systems are predetermined by the organization’s communication structures. Ruth Malan translated Conway's Law into a modern version in 2008 and looked at it from a different perspective. She stated, "If the architecture of the system and the architecture of the organization are at odds, the architecture of the organization wins."

So, the team structures and the communication between teams provide the foundation for successful product development. The team structure already gives us the first indication of whether a project will possibly be successful or could even be headed for failure if, for example, the system's architecture, the teams try to fight against the organization's architecture. The organization is compelled to produce designs that reflect or imitate the organization's real, on-site communication structure. This has significant strategic implications for any organization that designs and builds software systems, whether internally or through third parties.

For example, going back to the issue of silos, let’s assume an organization is arranged in functional silos including Quality Assurance, Database Administration, and Information Security. It is very unlikely that this organization will ever produce a software system with an underlying well-designed architecture for continuous flow without major handoffs between these silos. Moreover, the long release cycles already seem to be a predefined condition.

Now, let’s consider another instance where our goal is to transform a monolithic application structure into a microservice architecture, as shown in Figure 3. The probability is high that we will not be able to create a modular service architecture with an organizational structure that fosters large co-located teams. In self-critical terms, it is a difficult (almost impossible) mission not to reintegrate communication channels from the real world into the system architecture.

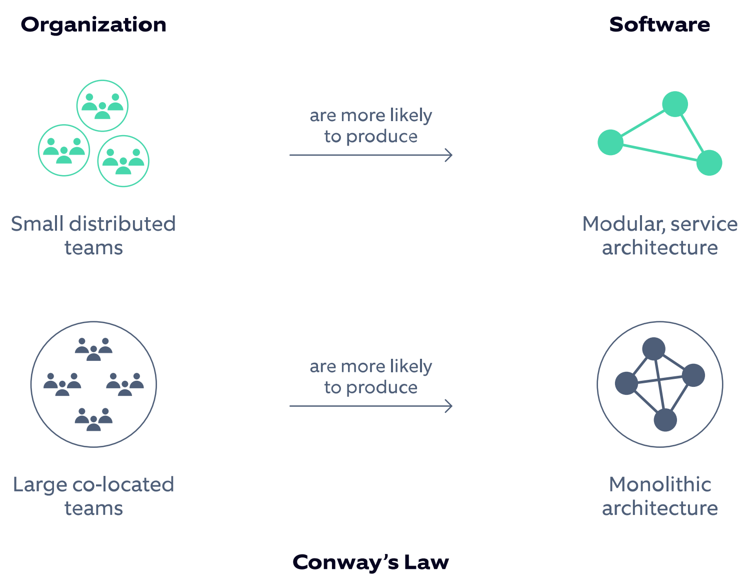


Figure 3: Conway's Law

Our observations

At Nagarro, we work in small agile units designed to bring an end-to-end mindset - called feature teams. These types of teams are entirely focused on the client's needs but always follow the DevOps mentality of "You build it, you run it, you own it."

Accepting and engaging the fact that such teams are often faced with challenges in the Cynefin's complex domain and acknowledging the homomorphic force often results in Conway's Law paving the way for more open discussions and identification of the most expedient solutions. Here, homomorphic means "having the same shape," so the organizational structure and system architecture remain in sync.

However, even as these two concepts help design an organization, they are only the first step towards a sustainable and continuous flow of value.

2.2 Identify Value Streams

Conway's Law tells us we always need to consider the desired system architecture, while assembling teams that will be responsible for delivering them. The technical expertise is required, not only in the development teams, but for designing organizational structures as well. It requires a high level of technical understanding to design an effective and adaptable organizational structure that supports our systems to evolve in the best way. So, if we want to develop a particular system architecture, appropriate team structures can help achieve the same. Whereas inappropriate structures can lead to undesirable outcomes that greatly differ from the original designs. But how much awareness can we attest to HR alone or other departments regarding design of software systems?

Deciding on the organization's team structure, responsibilities, and boundaries without input from technical leaders can prove to be greatly ineffective and irresponsible. The organization design and software design should be considered equally.

Good practices: Identify the best structure for your use case

- Using the technical knowledge from software development and software or solution architecture, we can derive some good practices and concepts to determine the right structure for our specific use case. These practices can help us find the value stream, the continuous flow of change.

- Software architecture that enables the business. It should be extensible but built on a common platform.

- Loose coupling helps to ensure that components do not have strong dependencies on other software parts.

- High cohesion results in components with well-defined and clear responsibilities but their internal elements are strongly bound to each other.

- Considering the end-to-end responsibility of cross-functional teams, it should support collaborative testing and version compatibility.

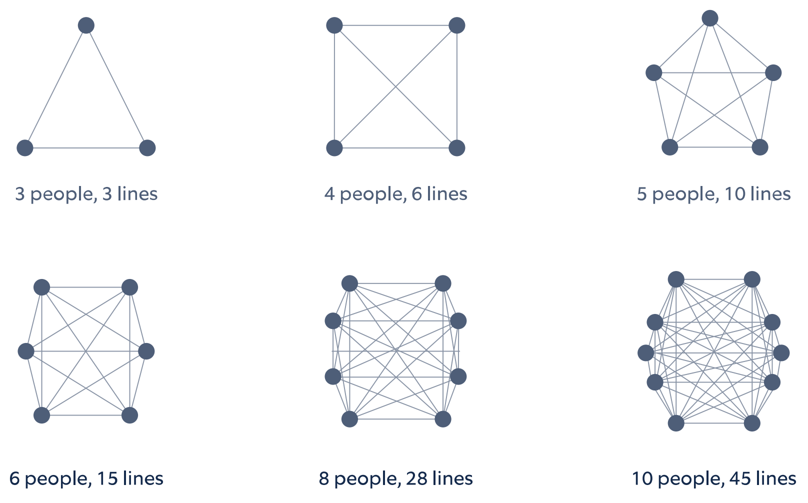
To better understand the boundaries of an organizational structure or architecture, **Context Mapping** is helpful.

It is a general-purpose technique, originating from the Domain Driven Design (DDD) toolkit. The technique helps architects and developers deal with various complications in software development projects. Context mapping can be applied to any type of scenario and provides a high-level view that helps us make strategic decisions.

It can help identify the so-called "Ubiquitous Language" in the program parts and clarify communication channels between them. The ubiquitous language should be the only language used to describe a model. Everyone on the team should be able to agree on any specific term without ambiguity, and no translation should be required. The maximum extent to which a model can be stretched without compromising its conceptual integrity is called context.

Through event storming, which is a workshop-based method for quickly finding out what is happening in the domain of the software program, a Context Map can be created. This in turn helps us identify the desired value streams. Since a change in team structures has a direct impact on communication lines between people and segments, tedious handovers and coordination must be taken seriously.

The organizational structure itself is linked to the architecture to be reached and cannot evolve beyond it. Fast deliveries to end users and release cycles are indirectly related to the communication channels. The more autonomous a team can work on its value stream, the more effective the system will be.



Communication lines

Fig. 4: Communication lines of people

You could also say the communication between effective teams, should be limited as much as possible to achieve the desired level of throughput. Unnecessary and unstructured communication becomes an overhead.

Our observations

Since we work with so-called feature teams at Nagarro, these teams are aligned with agile Scrum teams and possess all the skills required to successfully complete their client project. These teams are not permanent, they change from client situation to client situation, but are consistent for as long as possible, as long-lived teams provide the most sustainable benefits. It allows us to be extremely flexible in responding to clients. Similar to Scrum teams, feature teams have an upper limit of team members. We usually keep them as small as possible - maximum 5-8 people strong.

2.3 Cognitive capacity as the primary limiting factor

In cognitive psychology, cognitive load theory describes different types of cognitive load in the process of knowledge acquisition. Established by John Sweller and Paul Chandler, the theory assumes that learning is associated with cognitive load and describes how learning can be made easier or more difficult. Learning itself is a critical limiting factor in software delivery.

The theory assumes that the capacity of working memory is finite and that only a certain amount of information can be retained. People have a limited velocity to unlearn and relearn. The pace of change cannot be forced, it can only be fostered. Embracing this assumption as well as the importance of teams brings us to the key factor that assigning responsibilities and designing organizational structures is a cognitive load on a team level.

Cognitive Load attributes a particularly important role in learning and knowledge acquisition to the working memory. The working memory is responsible for problem-solving and information-processing mechanisms. There are three different kinds of cognitive load, defined by Sweller:

- **Intrinsic cognitive load** refers to the task that is caused by the task itself or depends on the difficulty and complexity. The more difficult and interconnected the task, the higher the intrinsic cognitive load.
- **Extraneous cognitive load** relates to the environment in which a task is executed and the design of its instructions.
- **Germane cognitive load** or learning-related load refers to efforts required to actually learn, understand, and excel certain concepts. Keeping intrinsic and extraneous cognitive load on the lower side makes room to focus on the germane load, leading to a more effective learning experience.

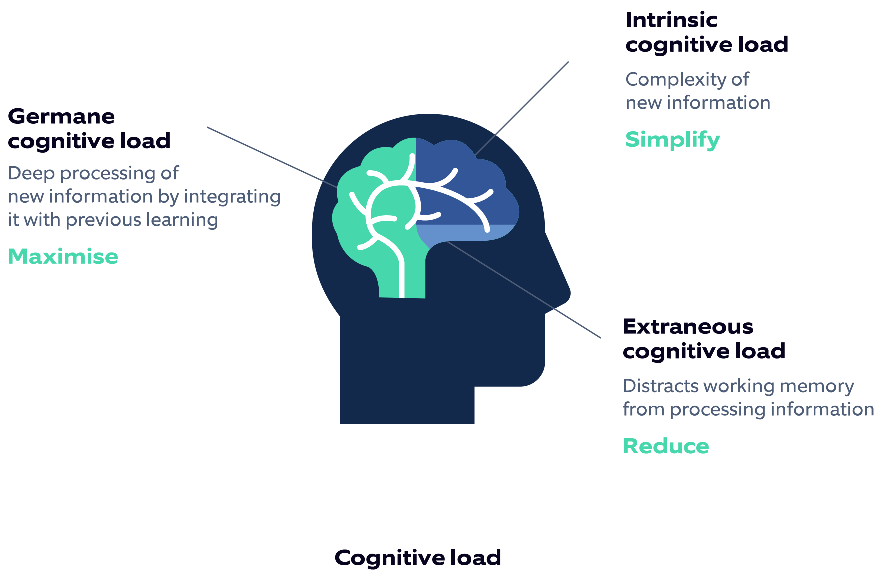

Intrinsic cognitive load
Complexity of new information
Simplify
Germane cognitive load
Deep processing of new information by integrating it with previous learning
Maximise
Extraneous cognitive load
Distracts working memory from processing information
Reduce
Cognitive load

Fig. 5: Different kinds of cognitive load

In terms of IT industry application, specifically to development teams, an intrinsic cognitive load for a delivery team is for example knowledge about the concept of Continuous Integration and how it can be applied. Extraneous cognitive load in software delivery is often related to knowledge of specific details that should be irrelevant to the task at hand. An example could be irrelevant details about instantiating a test environment that requires the user to execute several complicated, non-automated console commands. Contrary to intrinsic cognitive load (which is generally immutable), extraneous cognitive load can and should be minimized as much as possible.

Germane cognitive load is often related to the effort of connecting and understanding how certain business domains work and how software solutions can be applied or design how various isolated services would communicate with each other.

Our observations

At Nagarro, we use a simple and quick method to assess the cognitive load of feature teams - we ask them how they feel - "Is the team able to respond effectively and in a timely manner to the required work?" This is not a scientific study, of course, but it gives us a quick insight into whether teams are feeling overloaded. We often add free text to the assessment that prompts the teams with an option to give us more insights. You will be amazed at how accurately individuals are able to rate their load.

We use the same method in a wide variety of client situations. If the assessment is negative, the organization must take the necessary steps to reduce the cognitive load. This ensures that the team can work effectively and proactively again. Along the way, it boosts the morale, increases employee satisfaction and motivation within the team, as members see more value and meaning in their work.

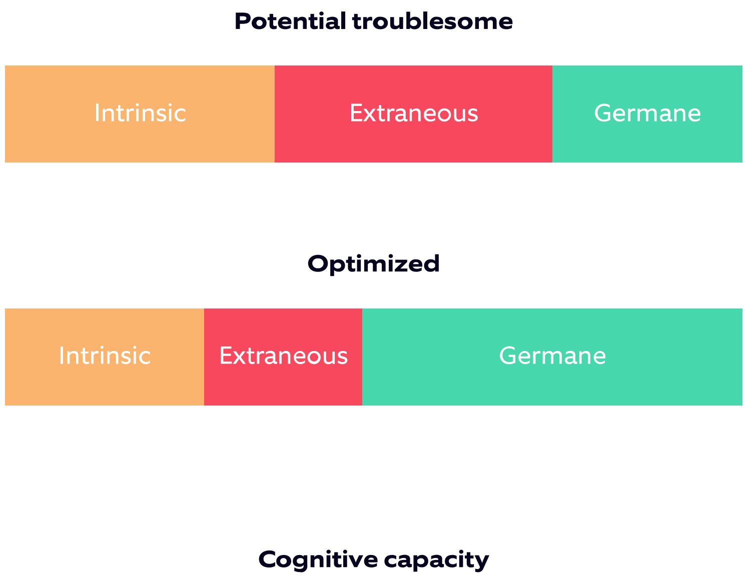


Fig. 6: Cognitive capacity

Identifying symptoms of overburdened teams with cognitive load is often relatively simple if organizations start actively looking for the warning signs. However, measures to mitigate it entirely is a complex problem to solve. In general, reducing extraneous cognitive load as much as possible is advisable. This is where automation (e.g., Infrastructure as Code, Test Automation, Continuous Integration & Delivery) and abstractions (e.g., Self-Service, establish common platforms) play an integral role. By removing the requirement to think about certain, irrelevant aspects of software delivery, the extraneous cognitive load can be minimized.

For the cognitive load of delivery teams, a profound way for an organization to help reduce the load is by investing in the development of employees, specifically those in development-related areas - developer experience (DevEx) becomes a key driver.

2.4 Foster developer experience (DevEx)

Foster a good developer experience to focus on the creation of continuous added value. When an organization strives for effective and high-performance delivery and operation of software systems, it is essential that the intrinsic and extrinsic cognitive load of teams are minimized. Thus, the cognitive capacity can be used to solve problems situated in the complex problem space.

Reducing the intrinsic cognitive load of a team or the team members is challenging since many tasks in modern software delivery projects are inherently complicated, but there are some techniques that can help minimize its impact. In the following, we highlight a small collection of possible practices that can help you reduce unnecessary cognitive load

- **Individual training programs and events:** At Nagarro, we offer a wide range of classroom trainings and self-learning via the highly successful Nagarro University (NagarroU). Additionally, it is also imperative to provide possibilities to share learnings, talk about successes or failures in projects or explore new ideas or current topics together (e.g., "Show & Tell" events). Offering those and other initiatives does not necessarily reduce the intrinsic cognitive load required to learn certain concepts. Still, it provides different perspectives on them, makes identifying important information more efficient, and is generally more enjoyable.

- **Practice pair programming or test-driven development:** We teach our colleagues to work directly on their project, their codebase to guarantee the success of the project and drive quality as a driver for innovation. We call this kind of learning experience - Technical Excellence Trainings.

- **Establish a technology radar:** Another area to consider is keeping an eye on a homogenous technology stack and ensure an appropriate selection of technologies depending on the specific context in which they should be applied. Especially in larger organizations, with multiple, independent teams and products, a technology radar is a valuable tool to reduce cognitive load caused by an unnecessary diverse technology landscape.

Extraneous cognitive load should be eliminated where possible and feasible. In development teams, it is often caused by redundant tasks or by commands that are not relevant for the task at hand and can be automated. It is important to consider that only extraneous aspects should be eliminated by e.g., automation, abstraction, or removal. However, to make good decisions relevant aspects should be as transparent as possible and not hidden under potentially unnecessary abstractions.

In case we realize that we lack certain skills or know-how in a specific area that is in demand with our clients, our colleagues have the possibility to create a specialized training program. The same has been done successfully in the field of Agile and DevOps for example. In these training programs, which are now also known as Shift-up, we focus on conveying the know-how to the employees in the shortest possible time but retaining the highest quality. This in turn helps us to spread the topic more widely and reach more colleagues – a win-win for everybody.

2.5 Enable teams

If we accept the premises outlined earlier, then a logical consequence is that to be effective, efficient, and sustainable, teams must be able to work autonomously and make decisions in the shortest possible time. A team must have the skillset it needs to do its job without the corresponding cognitive overload. It is as important to have a holistic view with the result in mind as it is to create and communicate a common goal.

Continuous learning must be encouraged. Sharing ideas for experiments, common interest, learning new ways of working that could make a team more efficient, or learning from other approaches is essential.

At Nagarro, we are 10,000 employees spread all over the world. Our mission is "Making distance irrelevant between intelligent people". Recently, we also introduced a "Work From Anywhere" #WFA policy

Therefore, connecting colleagues is not an easy task. Especially with DevOps, the topic of **breaking down silos** is communicated very actively. It is important to us to promote communication between as many parties as possible, always bearing in mind that too many lines of communication are not advantageous, in many cases even harmful.

Moreover, we have had very good experiences with Community of Practices (CoP) and other community forms to make the communication as substantial as possible and to prevent the flood of information from becoming too large and complex. These are not only forms that promote communication among peers, but these groups provide a great deal of know-how and **bring back innovation into the business**, which in turn can be incorporated into the organization's strategy.

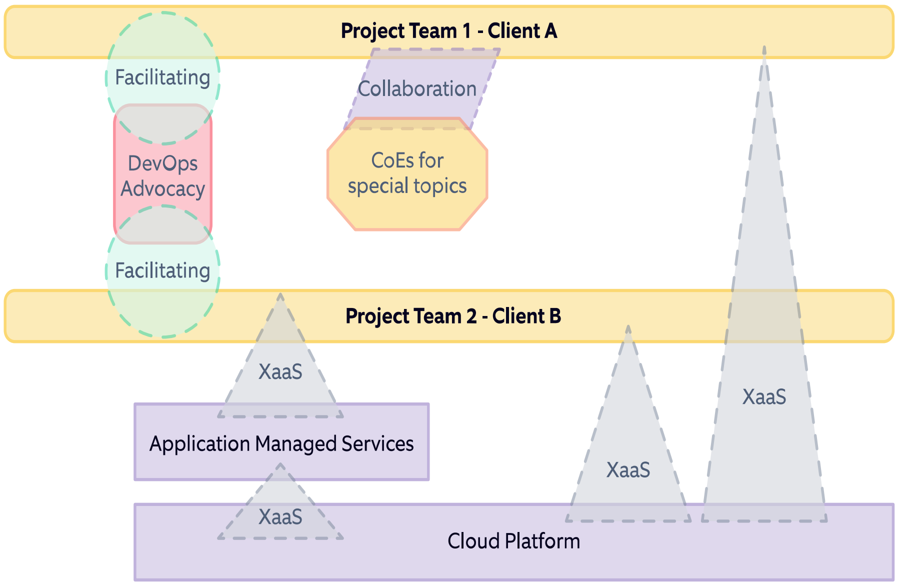
Among other things, we have global DevOps communities that currently meet virtually, every 2-4 weeks for about 2 hours. The format is up to the community. Current topics are discussed, colleagues show the latest findings from their projects, or problems are discussed for which the community may already be able to offer a solution. Other communities come together for ensemble programming or work on katas to foster their technical excellence.

In addition to the classic CoP, we also have other exchange formats, such as the above-mentioned “Show & Tell” or our “Global DevOps Meet”, which complete our learning journey. And we are already looking forward to meeting again during on-site events. There will be a major DevOps Summit within Nagarro, where people from all over the world come together to talk about lean, agile and DevOps oriented ways of working. Another measure that we consider a very valuable investment for our future are the so-called **Advocacy Teams**.

Phases	Starting point	Possible improvements	Status quo
Plan	1	1	1
Code	3	3	3
Build	2	2	3
Test	2	3	3
Release	2	3	3
Deploy	2	2	2
Operate	1	1	1
Feedback	N/A	N/A	1
Quality Devops stamp	No	Yes	Yes

DevOps Advocacy real-life example

Fig. 7: Example outcomes of an actual project where a Quality DevOps - Advocacy Team supported a feature team to improve in various areas. Improvements that were covered include implementation and improvements of CI-pipelines, automated provisioning of cloud-infrastructure as well as process improvements to amplify feedback loops.



Topology of DevOps Advocacy Teams

*Fig. 8: High level overview of how DevOps Advocacy Teams are integrated at Nagarro*their technical excellence.

Their purpose is to work with our feature teams and guide them on their personal journey. You could also call them Enablement Teams. We invest in the quality of our projects, but also in our people and in the development of their capabilities. Enablement teams take care of specific aspects to improve the way of working and technical excellence or DevEx. Let's give a view of how this area of enablement might change in the future. At present, the DevOps Advocacy Teams are small self-managed teams working in Business Units. In the future, we envision larger Centers of Enablement (CoE) whose primary goal would be to take the way of working to a new level.

Conclusion

Embracing DevOps entails more than following certain technical practices and principles. At Nagarro we try to consider DevOps as an encompassing concept influencing many decisions and areas that are often forgotten. While we firmly believe that there is no generically applicable blueprint or one-size-fits-all universally valid solution to the several challenges, we have tried to condense a framework in form of this guiding principles that are simple to understand and apply. And based on our experiments and experience in the past, these shall provide valuable results in many different domains, industries, and organizational contexts.

We don't do DevOps so that we can say "we do DevOps". We use lean, agile, and DevOps practices to deliver better value, faster, safer, and happier. Our focus is on continuously improve the outcome. So, we also recognize that this will not be the end state and we need to constantly adapt, improve, and rethink.

References

Patrick Debois: https://itrevolution.com/faculty/patrick-debois/

Forrester - 2018 The Year of Enterprise DevOps: https://go.forrester.com/blogs/2018-the-year-of-enterprise-devops/

Matthew Skelton & Manuel Pais - Team Topologies: https://teamtopologies.com/

Matthew Skelton & Manuel Pais - DevOps Topologies: https://web.devopstopologies.com/

Cynefin Framework: https://en.wikipedia.org/wiki/Cynefin_framework

Conway's Law: https://en.wikipedia.org/wiki/Conway%27s_law

Cognitive load: https://en.wikipedia.org/wiki/Cognitive_load

About the Authors

Jürgen Pointinger
Jürgen Pointinger is a DevOps enthusiast and Practice Lead at Nagarro. He supports clients in reaching the greatest sustainable, fast, and secure benefits. Thereby he emphasizes an open, value-adding culture, trust, and continuous learning. He started his career as a Developer before working as a Software/ Solution Architect, Team Lead, CTO, IT Consultant and DevOps Coach. He spends his free time with his family and learns from his children to look at the world with different eyes.

Stefan Gwihs
Stefan Gwihs is a Test Automation Architect and DevOps Coach at Nagarro. After completing his bachelor's degree in computer science and subsequently a master's degree in multimedia and software development, Stefan focused primarily on agile software development and test automation. He is certified by ISTQB® and IREB and has a very rich experience in the optimal handling of IT projects thanks to his diverse project activities.

Marketing Team

Editor
Deeksha Mamtani

Designer
Harsh Magan