Edition #3

Socio-Technical Architecture

The Essential Report for Technology Decision Makers

Foreword

In today's professional world, it's no longer enough to simply react to technological change — we must actively shape it. Time and again, we see that organizations and software architectures cannot be viewed in isolation; they are deeply intertwined and influence each other in complex ways. This is where the concept of socio-technical architectures comes into play.

Socio-technical architectures provide a framework for aligning technical systems with human collaboration in an optimal way. In a world that is becoming faster and more complex, they help maintain clarity while enabling flexibility and adaptability. The focus is not just on adopting new technologies, but also on integrating them effectively into existing organizational structures.

Sometimes, this even requires organizations to evolve in order to generate lasting value. While the approaches presented in this briefing originate in the IT field, their impact extends far beyond it. Whether you are leading a small team or managing an entire organization, the principles of socio-technical architectures can help you rethink work processes and empower employees with the tools they need to succeed in their roles.

This Technology Briefing offers insights into key concepts such as Team Topologies and Domain-Driven Design and demonstrates how they can be practically implemented. Our authors aim not only to provide a solid theoretical foundation but also to offer hands-on guidance for applying these principles to your specific challenges.

We firmly believe that the deliberate integration of system architecture and organizational structure is a critical competitive advantage. With this briefing, we hope to inspire you to future-proof your organization by actively shaping its socio-technical architecture. Striking the right balance between teams and their software responsibilities is essential — when done well, it can boost efficiency while also fostering a better working environment.

Martina Freers
Principal Consultant, Geschäftsleiterin

INNOQ TECHNOLOGY BRIEFING

Intro

Common Approaches in the Field of Socio-Technical Architectures

In the age of digitalization and modernization, companies face the challenge of not only keeping up with technological advancements but also optimizing their organizational structures. Socio-technical architectures play a crucial role in this process by focusing on the flow of work, team collaboration, and the seamless integration of technical and social systems. This article explores two key approaches — Team Topologies and Domain-Driven Design (DDD) — and demonstrates how they can help organizations take the first steps toward consciously designing their socio-technical architecture.

Iterative Adaptation for Adaptive Organizations

Organizations are not static systems — they are adaptive networks that continuously evolve. Working within the realm of socio-technical architectures must therefore be an iterative and ongoing process. There is no such thing as "done"; instead, change requires regular adjustments and feedback loops. Two fundamental principles are essential in this context:

Psychological Safety: Teams need an environment where they can freely experiment, learn, and make mistakes without fear of negative consequences. A culture of trust fosters innovation and continuous improvement.

Fast Flow of Work: In a world driven by Economies of Speed, it is crucial to identify and eliminate bottlenecks in workflows. Reducing friction in processes allows organizations to adapt quickly and remain competitive.

Team Topologies: Structuring for Fast Flow

The Team Topologies approach, introduced by Matthew Skelton and Manuel Pais in their 2019 book of the same name, provides a clear framework for organizing teams around value streams and workflows. The goal of Team Topologies is to reduce cognitive load on teams while maintaining a strong focus on the flow of work. This approach is particularly relevant for IT organizations that must navigate increasing complexity and speed. By structuring teams in a way that aligns with their responsibilities and capabilities, companies can improve efficiency, reduce friction, and create an environment where teams can deliver value more effectively.

Core Concepts of Team Topologies

Team Topologies defines four fundamental team types, each playing a distinct role in optimizing workflow and delivery:

Stream-Aligned Teams: These teams contribute directly to value creation for customers. They own end-to-end delivery for a specific product, service, or business capability. All other team types exist to support Stream-Aligned Teams, enabling them to deliver as efficiently as possible.

Enabling Teams: These teams help Stream-Aligned Teams develop new capabilities by providing guidance and expertise in specific technical or business domains. They focus on knowledge sharing and mentorship, rather than owning long-term delivery themselves.

Complicated Subsystem Teams: These teams handle highly complex technical components that require specialized expertise — often in areas where hiring skilled professionals is particularly challenging. Their work involves deep technical problem-solving that cannot be easily distributed across other teams.

Platform Teams: These teams create self-service platforms that enable other teams to build and deliver value more effectively. By providing reusable infrastructure, tools, and services, Platform Teams reduce cognitive load for Stream-Aligned Teams and help accelerate delivery.

Beyond the four team types, Team Topologies introduces three modes of team interaction: teams work collaboratively when needed (Collaboration), provide services with minimal coordination effort (X-as-a-Service), and offer temporary support to other teams (Facilitating). However, reducing Team Topologies to just these four team types and three interaction modes would be misleading. At its core, the approach is about organizational strategies that establish fast flow within delivery organizations. To achieve this, organizations must regularly assess and refine their structures. Techniques like Value Stream Mapping and Flow Engineering can be valuable tools in this process.

Another essential aspect of Team Topologies is its emphasis on cognitive load. Only teams with a manageable cognitive burden can deliver quickly and efficiently. Overloaded teams struggle with complexity, which leads to slower delivery and inefficiencies.

Rather than being a rigid framework, Team Topologies is an iterative approach. Decision-makers can apply it collaboratively with their teams to adapt more quickly to market changes and optimize the flow of work within their organizations.

Domain-driven Design: Purpose Through Clear Domain Boundaries

Domain-Driven Design (DDD) was introduced in 2003 by Eric Evans in his book of the same name. The approach focuses on business-driven modeling of complex system landscapes. Although its origins lie in software architecture and development, DDD has evolved significantly beyond that. The core idea of DDD is to manage complexity in modern organizations by dividing them into well-defined domains based on business needs. These domains are then structured using Bounded Contexts, which ensure that each team operates within a clearly defined and coherent scope. Team Topologies explicitly recommends aligning team boundaries with Bounded Contexts. This helps organizations reduce dependencies, improve clarity, and ensure that teams can work effectively within their respective domains.

Alignment on Purpose

A central aspect of Domain-Driven Design (DDD) is aligning teams with purpose within their Bounded Contexts. These contexts do more than just define technical boundaries — they also establish clear responsibilities and focus areas for teams. This concept aligns closely with the principles from Daniel H. Pink's book Drive, which identifies three key elements of motivation: Autonomy, Mastery, and Purpose. DDD provides a foundation that enables teams to achieve these elements in their work:

Autonomy: Clearly defined boundaries foster team ownership and responsibility.

Mastery: Within a Bounded Context, teams can specialize in a specific problem domain, deepening their expertise.

Purpose: A strong domain focus allows teams to clearly understand how their work contributes to the organization's overall strategy.

Strategic Staffing and Procurement: Core, Supporting, and Generic Domains

Domain-Driven Design (DDD) distinguishes between Core Domains, Supporting Domains, and Generic Domains, providing a structured approach to strategic staffing and procurement decisions:

Core Domains: These are business-critical and provide the greatest competitive advantage. They should typically be developed and maintained in-house to ensure long-term expertise and innovation.

Supporting Domains: These enable core processes but are less critical to competitive differentiation. In these areas, partnerships or external support may be a viable option.

Generic Domains: These are standardized areas that do not provide a competitive advantage. They should be outsourced or purchased whenever possible to optimize efficiency.

This classification helps IT decision-makers make strategic make-or-buy decisions, ensuring that internal expertise is focused on key business areas while supporting and generic functions are efficiently outsourced.

Iteration and Customization: No Silver Bullet

All the approaches presented offer valuable tools, but none of them are one-size-fits-all solutions. Decision-makers must adapt each method to the specific needs of their organization. An iterative approach is essential, as organizations are dynamic and their structures must be continuously assessed and optimized.

The values of the Agile Manifesto serve as guiding principles in this process. Statements like "Responding to change over following a plan" and "Collaboration over contract negotiation" emphasize that people and teamwork must be at the heart of any transformation.

Conclusion: Opportunities for SMEs and Enterprises

A socio-technical perspective on the interactions between domains, architectures, and organizations provides IT decision-makers with a strong foundation for making their organizations more agile, efficient, and future-proof. By combining methods like Team Topologies and Domain-Driven Design, companies can not only enable fast flow but also make strategically sound decisions regarding staffing and technology.

These approaches go beyond technical excellence — they create an environment where teams can experience Autonomy, Mastery, and Purpose. This is a key factor in attracting and retaining top tech talent over the long term. Ultimately, success lies in iterative adaptation and a clear focus on the unique needs and goals of each organization.

Michael Plöd
INNOQ

Platforms, Teams, and APIs

How Do They Fit Together?

One of the major trends in recent years has been **Platform Engineering**, as many organizations strive for more efficient and faster development practices. The term has gained traction since 2018, particularly following an influential article by Evan Bottcher[1]. Looking at the origins of this concept provides valuable insights into the relevance of Platform Engineering today and its potential future developments.

From Pages to Services

The early 2000s marked a turning point with the rise of the web. Tim O'Reilly, publisher and founder of a well-known media company, was one of the first to see the web not just as a human-centric system (where web pages are delivered to browsers) but also as a global service platform that enables communication between programs.

He shared this perspective in early 2000 during conversations with Jeff Bezos, then CEO of Amazon. At the time, Amazon was a rapidly growing online marketplace. Bezos was inspired by the idea that the future lay in making businesses faster and more flexible — and that services were the right building blocks to achieve this. Bezos responded swiftly and decisively. In 2002, he issued the now-famous "Bezos Mandate."

Jeff Bezos made a clear and bold directive: from that point forward, all teams at Amazon had to offer their services via programmable interfaces (APIs). These interfaces needed to be designed not just for internal use but also for external consumers. The goal was to reduce tight coupling between components and make all services easily discoverable and reusable.

The mandate did not prescribe a specific technology for these interfaces — as long as they were network-based.

The motivation behind this decision was clear: as a fast-growing, ever-evolving marketplace, Amazon required an IT culture and structure that prioritized speed and flexibility. Traditional software development challenges — such as long development cycles and high coordination overhead between teams — needed to be addressed.

The Bezos Mandate was both simple and radical, transforming Amazon's software architecture with the ultimate goal of accelerating value creation.

As a result of this mandate, Amazon launched its first e-commerce APIs in 2004, which helped drive the growth of its marketplace. However, the most influential outcome came in 2006 with the public release of Amazon Web Services (AWS) APIs, opening entirely new business opportunities.

But what exactly did the mandate say that had such a profound impact on Amazon's success?

Every Team Has an API

To enable Amazon teams to develop as quickly as possible, dependencies on other teams had to be minimized. This was achieved by requiring every team to provide an API (the original mandate used the term "service" instead). The mandate was brief but had far-reaching implications. It consisted of the following key points:

1. All teams must expose their data and functionality through service interfaces.
2. Teams must communicate exclusively through these service interfaces.
3. No other forms of communication are allowed — no direct linking, no direct database access to another team's storage, no shared memory, and no backdoors. The only permitted communication is through service calls over the network.
4. The choice of technology does not matter — HTTP, CORBA, Pub/Sub, custom protocols — anything is allowed.
5. All service interfaces must be designed from the ground up to be externalizable. Teams must plan and architect their interfaces as if they were intended for external developers. No exceptions.

At the time, this was a radical approach that significantly improved the scalability of team interactions compared to the tightly integrated systems that were common back then. When viewed through this lens, it becomes clear that the Bezos Mandate was primarily about minimizing inter-team dependencies (loose coupling) and restricting team interactions to a well-defined communication channel.

The mandate also had far-reaching consequences. Steve Yegge described them in his famous "Google Platforms Rant" from 2011[2], where he reflected on his experiences at Amazon. His insights reveal that implementing the Bezos Mandate was complex and took years. In an API-driven company, even fundamental aspects — such as debugging issues or locating services — change dramatically.

However, this discussion is not just about the technical complexity of decentralized software architectures. More importantly, it highlights how team structures and technical structures are inherently interconnected.

How Teams Interact

The Bezos Mandate fundamentally changed how teams interact by establishing loose coupling as the default — and in fact, the only permitted — approach. However, this shift did not automatically make collaboration between teams easier or less complex.

Since the 2019 publication of the book Team Topologies[3], a new perspective on team interactions has gained traction. One of the key ideas introduced is the concept of a "Team API." Unlike the Bezos Mandate, which focused solely on technical APIs, the Team API concept extends beyond just interfaces and services. It also includes practices, collaboration patterns, and the key people involved in a team's processes. The idea is that loose coupling should be complemented by additional information that helps external teams understand how to interact with a given team effectively.

This approach further strengthens team autonomy, as teams operate externally as service providers. However, with greater autonomy comes increased responsibility — teams must now handle the entire development lifecycle independently.

To mitigate this natural but unintended consequence of increased autonomy, the platform model has emerged as a solution. This includes not only traditional infrastructure platforms but also a more modern evolution: the Internal Developer Platform (IDP), which provides streamlined tools and services to help teams maintain their autonomy while reducing operational overhead.

Team Efficiency and Platforms

The fundamental difference between an infrastructure platform and an Internal Developer Platform (IDP) lies in their composition and purpose. While an infrastructure platform consists of standard market components (such as runtime environments and programming languages), an IDP is specifically tailored to the needs and practices of an organization.

This distinction is why Team Topologies introduces the concept of Platform Groups. These teams are responsible for relieving other teams of repetitive and potentially time-consuming tasks by implementing them within the IDP. As a result, regular teams (referred to as Stream-aligned Teams in Team Topologies) can work more efficiently by offloading certain responsibilities to the platform.

From a Team Topologies perspective, this approach achieves two key objectives: First, it creates the "Logical Platform," which is perceived and consumed as a cohesive platform. This enables teams to develop more easily by leveraging the platform's capabilities. Second, "Platform Groups" collectively contribute to the provision and continuous development of the Logical Platform — while potentially operating independently from one another.

Another advantage of the platform model is that it facilitates and encourages component standardization. For example, if teams previously used different monitoring solutions, a platform can offer monitoring as a standardized service, ensuring consistency across the organization.

Platform Engineering Supports Teams

The term Platform Engineering refers to a socio-technical approach that leverages team structure and responsibilities within the development process to reduce cognitive load and increase efficiency.

The growing popularity of Platform Engineering and platform teams today is primarily driven by two factors. First, the rapid increase in digital products has made efficiency gains in development more impactful than ever. Second, the demand for faster development cycles has intensified, requiring organizations to accelerate how they build and deploy digital products and solutions.

When development becomes simpler and faster (and therefore more cost-effective), companies can pursue more exploratory strategies — which leads us to the final missing piece in the modern platform puzzle.

Team Efficiency and Effectiveness

Platform Engineering primarily focuses on improving the efficiency of the development process. While this is both necessary and valuable, it does not automatically ensure that the right things are being developed.

This is where the concept of optionality comes into play, which Stephen Fishman and Matt McLarty explore in detail in their recent book[4]. Optionality means strategically maintaining as many options as possible to maximize value creation within an organization, while continuously refining and optimizing this process.

To achieve this, Fishman and McLarty argue that an organization's default approach should be to enable the reuse of business serviceswherever possible. This brings us back full circle to the Bezos Mandate, which was introduced earlier in this article.

Although the Bezos Mandate is frequently cited, its radical implementation is rare. This is partly because it is not always the most cost-effective approach for a given business model at a particular moment. Instead, it represents an investment in decoupling services, thereby creating greater flexibility for future developments. This concept, known as "Unbundling", is described by Matt McLarty as follows:

"What is it about APIs? They're decontextualized business capabilities. That's what unbundling is focusing on, is how do you decontextualize things... It's the fact that capabilities can be decontextualized, and then recontextualized in different ways, that makes them so valuable."

Unbundling requires effort and resources, and its strategic logic is not always appropriate in every scenario. Similarly, loose coupling comes with costs. However, these investments contribute to resilience and flexibility — values that are becoming increasingly critical in today's rapidly evolving world.

That does not mean that business and IT need to be completely overhauled. Instead, it calls for a shift in focus — toward teams, their interactions, and the potential of these structures. Team Topologies addresses this with the concept of the "Thinnest Viable Platform." This approach does not require a radical restart but instead aims to provide a minimalistic platform that offers only the essential functions needed to support product teams, simplify their work, and avoid unnecessary complexity.

This trend is also evident in the API product market, which plays a key role in supporting unbundling within organizations. After a period of stagnation, momentum in this area has surged. New vendors are emerging with API portals and marketplaces, and companies are increasingly making strategic investments in API strategies and programs. While part of this shift is undoubtedly driven by the rise of AI applications that rely on APIs, the trend toward strategic API management actually began well before the current AI boom.

Outlook

It is likely that in the future, we will see team structures contributing not only to greater efficiency but also to higher effectiveness and a more flexible strategic alignment at the business level. In summary, we can observe the following key developments being increasingly adopted by organizations:

Teams: Dependencies are being loosened, allowing teams to work faster.
Platforms: IDPs help teams operate more efficiently by reducing overhead.
APIs: APIs enable organizations to evaluate and implement more options, increasing the effectiveness of what teams deliver.

An interesting aspect of this dynamic is that technology itself plays a relatively limited role at this level. Of course, loose coupling must be implemented properly from a technical perspective — through good API design and reliable versioning models, for example. However, in the end, these technical elements are merely gears in the larger socio-technical system of an organization, which remains in a continuous search for value creation.

Erik Wilde
INNOQ Schweiz

[1] Evan Bottcher, "What I Talk About When I Talk About Platforms", March 2018. https://martinfowler.com/articles/talk-about-platforms.html
[2] Steve Yegge, „Stevey's Google Platforms Rant", 2011. https://gist.github.com/chitchcock/1281611
[3] Matthew Skelton und Manuel Pais, "Team Topologies", IT Revolution, september 2019.
[4] Stephen Fishman und Matt McLarty, "Unbundling the Enterprise: APIs, Optionality, and the Science of Happy Accidents", IT Revolution, September 2024

Socio-Technical Architecture as a Competitive Advantage

In today's business world, technological progress is not just about increasing efficiency — it also unlocks entirely new business opportunities that would be unthinkable without these technologies. As a result, leveraging new technologies effectively has become a key competitive advantage.

However, IT decision-makers are increasingly realizing that technology alone is not enough. Instead, companies must understand themselves as socio-technical systems, where technical systems and human interactions are deeply interconnected.

This article explores why socio-technical architectures matter for IT decision-makers and how they can create a competitive edge for organizations.

Why Socio-Technical Architectures Are Key to Competitiveness

Speed and Agility

Digital transformation is accelerating the need for organizations to adapt quickly to market changes. At the same time, knowledge about customer needs, regulations, budgets, technologies, and processes is becoming more widely distributed across an organization. This distribution allows specialists to optimize each dimension of the value chain independently.

However, when knowledge is fragmented across many teams, it becomes much harder to analyze and optimize processes. Before any improvements can be made, all relevant insights must first be brought together into a cohesive picture — a challenge that slows down decision-making.

A socio-technical architecture takes a holistic approach to these challenges by balancing different perspectives and making them transparent in organizational structures. Having a real-time overview of all key dimensions of the value chain enables organizations to assess, discuss, and adapt their processes more quickly.

Companies that adopt a socio-technical perspective are better equipped to respond to shifting customer demands, regulatory changes, and market trends. As a result, they can bring innovative products to market faster than their competitors.

Improved Collaboration and Employee Satisfaction

Socio-technical architectures thrive when employees engage in close collaboration and are supported by tools that facilitate communication and knowledge sharing across teams.

This kind of exchange enables employees to learn from one another across departments and co-develop innovative solutions that align with business goals. When employees see their contributions as essential parts of a broader value chain and collaborate with colleagues from different areas toward shared objectives, it enhances job satisfaction and strengthens employee retention. By fostering a work environment that values cross-team collaboration, organizations not only improve team effectiveness but also reduce costs associated with employee turnover — a major factor in long-term business success.

Tailored Technology

Since socio-technical architecture places people at the center of the solution, their needs and requirements are given greater consideration. This results in customized solutions that optimally support value creation within business units. By focusing on relevant, business-driven functionality, organizations increase efficiency while minimizing time spent on non-value-adding activities.

Implementing Socio-Technical Architectures

To actively shape socio-technical architectures, all relevant dimensions must be considered and integrated into a cohesive picture. This requires both gathering diverse information and requirements and analyzing their relationships to understand how different areas interact. This process can be applied organization-wide or to specific segments — ideally both, with responsibilities distributed across multiple teams.

Collecting and Documenting Information

Keeping all dimensions of an organization up to date and comprehensively documented — while maintaining relationships between them — is a time-consuming effort. This documentation often competes with operational demands and ongoing development work.

Many Enterprise Architecture (EA) initiatives fail because they attempt to fully document the current state of an organization beforedefining a target state. This requires extensive interviews and alignment meetings, which delay value creation. A more effective approach is to establish standardized documentation practices, introduce shared tooling, and prioritize data collection in areas experiencing the most pressure for change.

The goal is to minimize the overhead associated with documentation and cross-team coordination without compromising operational efficiency. At the same time, enough structured communication and documentation must exist to provide a sufficiently clear overviewfor fact-based decision-making.

A key principle in documentation is always answering the question: "For what purpose?" Clearly defining the objective of a process or technical implementation allows teams to assess its actual contribution to value creation.

Roles and Interactions

Enterprise Architects derive insights from business goals and corporate strategy to determine which business capabilities should be supported by specific IT capabilities and initiatives. In essence, they translate leadership objectives into technical and organizational requirements. Requirements Engineers and Business Analysts gather detailed business needs from different departments. Their work aligns with overall business goals but operates at a much finer level of detail than traditional Enterprise Architecture artifacts. Solution Architects use these strategic and operational insights to design products and solutions that align with both the company's overarching objectives and the specific needs of business units.

At all levels, architects involve key stakeholders in planning efforts — not just for technology decisions but also to identify necessary process and organizational changes.

Many companies have additional architecture roles, such as Domain Architects and Software Architects. While they all balance different stakeholder perspectives and develop solutions accordingly, they operate at different levels of abstraction.

Together, these different architectural layers form the complete socio-technical architecture of a solution.

Continuous Improvement

Socio-technical architectures are not static structures — they should be regularly reviewed and adapted to meet new requirements. By establishing a continuous improvement process, including regular feedback loops and optimizations, organizations can ensure the long-term success of their architecture and create a sustainable competitive advantage.

A Success Story from Practice

One of the most well-known examples of a company that has aligned its technology development with its organizational structure is Spotify.

Spotify's Squads, Tribes, Chapters, and Guilds model was designed to manage the complexity of its technical solutions while optimizing collaboration across teams.

This holistic approach proved to be so effective that the Spotify model has since become a template for many other companies.

However, what is often overlooked is that Spotify's model was developed during a time when the company was significantly smaller than it is today. The original organizational approach did not yet address the challenges of large-scale development environments. Consequently, Spotify has continuously evolved its own organizational structure to better align technology and collaboration modelswith its growing scale and changing needs.

Conclusion

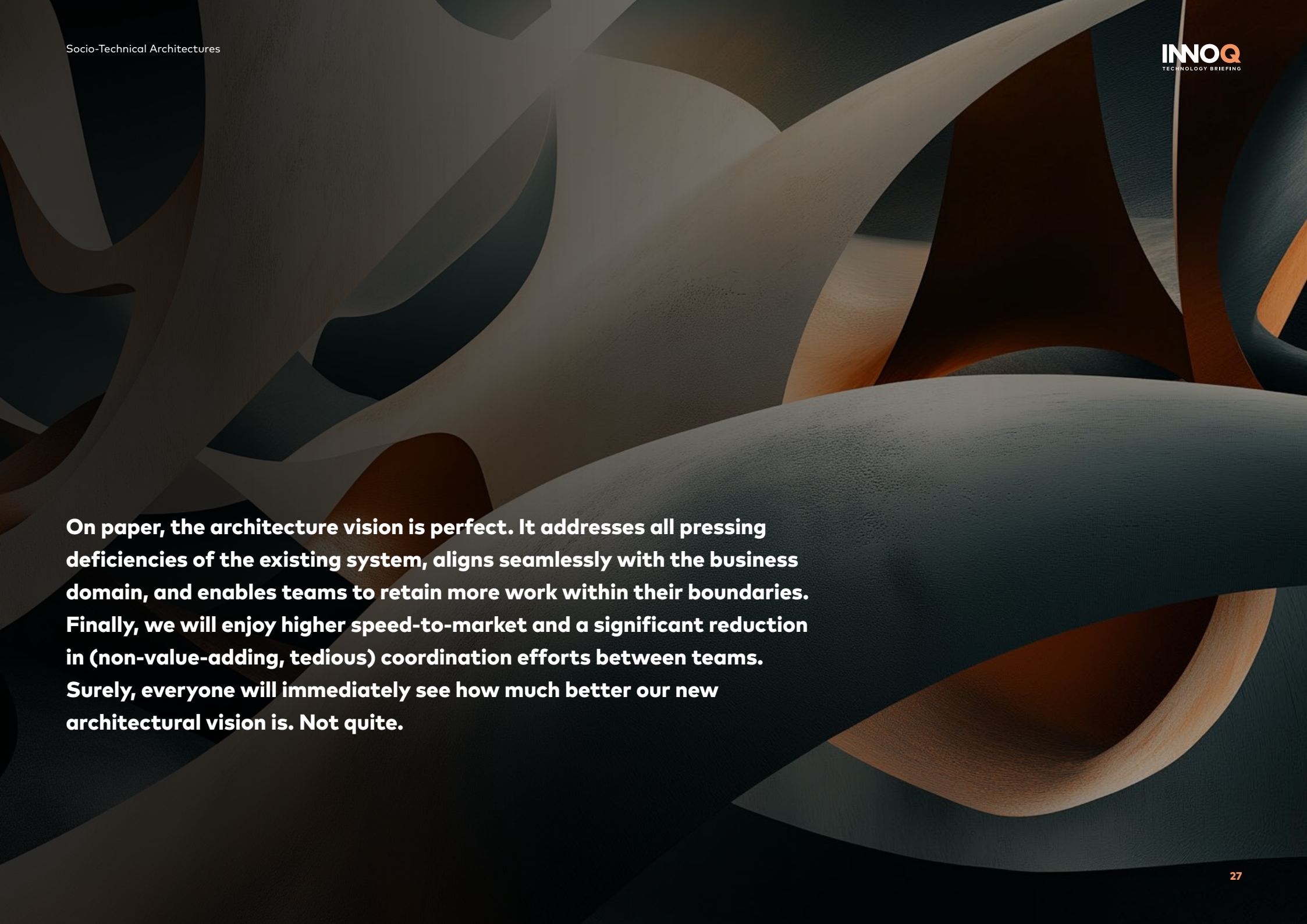
Socio-technical architectures are far more than just a trend. They provide IT decision-makers with a powerful approach to integrating technology and people in a way that makes organizations faster, more agile, and more innovative.

By adopting a socio-technical architecture, companies can not only enhance employee productivity and satisfaction but also significantly strengthen their long-term competitiveness.

For organizations aiming to succeed in the digital era, implementing socio-technical architectures is not just an option — it is a critical investment in the future.

André Aulich
INNOQ

Don't Forget the People – When Architectural Changes Become a Change Project



On paper, the architecture vision is perfect. It addresses all pressing deficiencies of the existing system, aligns seamlessly with the business domain, and enables teams to retain more work within their boundaries. Finally, we will enjoy higher speed-to-market and a significant reduction in (non-value-adding, tedious) coordination efforts between teams. Surely, everyone will immediately see how much better our new architectural vision is. Not quite.

New Architecture, New Skills

A major aspect of architectural transformation is that introducing new architectures often requires new skills and expertise. Suddenly, technologies, frameworks, integration patterns, or runtime environments become relevant — many of which are completely unfamiliar to team members. This learning curve should not be underestimated, and the resulting uncertainty can quickly become a burden. Simply offering a few training sessions is not enough — organizations need a sustainable learning culture that allows room for trial and error while ensuring support from experienced colleagues.

Breaking Down Silos – Easier Said Than Done

One of the major promises of modern software architectures is the elimination of old silos. Teams are expected to work cross-functionally and take ownership of their domain. However, this cultural shift does not happen automatically.

Someone who has worked exclusively on backend development may struggle when suddenly required to dive into frontend technologies — and vice versa. Similarly, taking on more responsibility can feel overwhelming for some. It is crucial to actively engage employees in this transition and to take their concerns seriously.

Communication Barriers in Focus

Aligning software architecture with organizational structures means that developers will need to collaborate more closely with business teams — often more than ever before. For example, defining subdomains within a company requires constant interaction between developers and domain experts.

This process frequently exposes communication barriers that may have previously gone unnoticed. Differences in terminology, expectations, and priorities can lead to misunderstandings. Roles that evolve significantly — such as business analysts — can create uncertainty or even feelings of status loss. At the same time, not all engineers are naturally inclined to immerse themselves deeply in business processes and objectives.

To address these challenges, organizations must proactively create spaces for constructive dialogue — whether through workshops, joint retrospectives, or mediators who help bridge the gap between technical and business teams.

Incrementalism Preferred – But Not Without Risks

In most cases, I advise clients to approach modernization incrementally rather than pursuing a massive "Big Bang" overhaul. From an architectural and business perspective, this approach makes a lot of sense:

Instead of committing large amounts of capital and capacity to a high-risk transformation, an incremental strategy allows for quick wins and course corrections when plans inevitably collide with reality.

However, the human factor must also be considered. Incremental modernization often involves creating small "Architecture Modernization Teams" that get early access to new technologies and approaches or work on a greenfield project.

For longstanding teams, this can fuel resentment or resistance toward new technologies and approaches. A strong modernization strategy should include mechanisms to actively counteract these effects and ensure broad team engagement.

The Challenge of Reorganization

Perhaps you have already achieved some early successes with the new architecture, and your modernization team is growing alongside the new software. A temporary matrix organization might suffice for initial experiments, but long-term success requires structural clarity.

If new architectural principles are taken seriously, they will sooner or later raise questions about organizational team structures. Ideally, the building blocks of a business-aligned software architecture should match team responsibilities — which often necessitates restructuring.

These deep organizational changes are rarely conflict-free. Successfully implementing them requires courage and sensitivity to ensure that employees are actively involved and that resistance is minimized.

Putting People at the Center

All these challenges highlight one fundamental truth: A new software architecture is not just a technical challenge — it is also a cultural and organizational change project.

It is not enough to simply define a new architecture and push for rapid implementation. Instead, the needs, fears, and expectations of employees must be considered from the very beginning. Communication, support, and active team involvement are essential to turning a vision into reality.

Because in the end, it is not just about building a better architecture — it is about creating an organization that grows through change and emerges stronger.

And to achieve that, we must put the people behind the code at the center of our efforts.

Johannes Seitz
INNOQ

How Much Thinking Can a Team Handle?

The Role of Cognitive Load in Socio-technical Systems

The term Cognitive Load has become a common reference point in recent discussions about team structures, roles, and task distribution in IT organizations. However, these discussions often fail to distinguish between the different purposes of IT organizations and frequently rely on an oversimplified concept of Cognitive Load.

It's reasonable that Cognitive Load is relevant in IT, particularly in software engineering. Software engineering is knowledge work performed primarily through thinking and communication — cognitive processes. Starting with the premise that high Cognitive Load is problematic, we can ask: How can Cognitive Load be reduced in an IT organization?

The Cognitive Load Theory

The concept of Cognitive Load, formulated by John Sweller and colleagues, addresses how much mental effort a person must expend to learn something. Sweller distinguishes three types of Cognitive Load that are fundamental to modern educational theory and can be applied to software engineering.

Intrinsic Cognitive Load represents the inherent complexity of a task: Adding two numbers is inherently less complex than performing polynomial interpolation. Consequently, the **Intrinsic Cognitive Load** for learning addition is lower than for learning polynomial interpolation. The **Intrinsic Cognitive Load** of a subject matter is inherent and cannot be altered.

However, it's possible to build up the **Intrinsic Cognitive Load** gradually through different presentation approaches. For example, elementary school children first learn to add numbers from 1 to 10, then from 1 to 100. After mastering these concepts, the entire number range is introduced, and addition is applied as a general principle.

In essence: The more aspects relevant to a subject, the higher the **Intrinsic Cognitive Load**.

This occurs because successful learning requires simultaneously holding these aspects in working memory to connect them. By appropriately structuring learning material, abstraction levels for coherent sub-aspects can be introduced, allowing learners to progress incrementally — but this doesn't reduce the actual **Intrinsic Cognitive Load** of the subject.

Extraneous Cognitive Load describes additional burdens created outside the learning subject. This stems not from the subject itself but from how the material is prepared and the environment in which learning takes place. This is particularly important for multimedia learning materials, as both movement and sounds consume working memory capacity and can therefore undermine the presentation described above. Examples include videos where the presenter is visible despite their visibility not contributing to understanding the content, or footnotes that contain relevant information but interrupt the reading flow.

Extraneous Cognitive Load can be managed relatively well, although reducing it often requires effort. A quiet environment, appropriate and consistent visualization, and a clear thread in the didactic structure are measures that can influence Extraneous Cognitive Load.

Finally, Germane Cognitive Load describes the effort of connecting elements in working memory with previously learned elements in long-term memory. Germane Cognitive Load essentially represents the "aha moment" when comprehension occurs. According to Sweller and colleagues, this comprehension becomes easier when Intrinsic and **Extraneous Cognitive Load** are lower. Learners can primarily only experience Germane Cognitive Load. Differentiating between Intrinsic and **Extraneous Cognitive Load** and working with these concepts is only possible for those who prepare learning materials.

Cognitive Load in Software Engineering

In software engineering, learning occurs with four specific subject areas: First, when learning a business domain; second, when understanding what a specific requirement means; third, when learning tools (programming languages, frameworks, etc.); and fourth, when comprehending unfamiliar code.

Each of these four learning areas carries an **Intrinsic Cognitive Load** according to the model presented above.

Business domains have their own complexity, which varies considerably. Depending on how many aspects of such a domain a software engineer has already mastered, learning a new one becomes easier.

Consequently, switching between similar business domains is more manageable: The remaining **Intrinsic Cognitive Load** when switching from balance sheet accounting to payroll accounting is lower than when switching to smart home control.
The **Intrinsic Cognitive Load** of the three domains doesn't change, but the Germane Cognitive Load for learners differs significantly.

New requirements also have an **Intrinsic Cognitive Load** and must be understood before implementation. Unlike learning the business domain, new requirements appear more frequently and in smaller pieces. Understanding the business domain can be considered a prerequisite for understanding the requirements:

If both the business domain and the requirement are new, the Germane Cognitive Load of both learning subjects increases, as they each function as **Extraneous Cognitive Load** for the other.

Learning new tools is a frequent necessity in software engineering. The impact on the **Intrinsic Cognitive Load** relevant to learners is influenced by familiarity with similar tools.
Thus, the remaining **Intrinsic Cognitive Load** when learning a new version of a familiar framework is generally lower than when learning a completely unfamiliar framework that essentially performs the same task.

On the other hand, certain tools cannot be learned in isolation: It's nearly impossible to learn a concept like Object-Oriented Programming without also learning a programming language.

The challenge in presenting these two learning subjects is to structure the learning path so that the **Intrinsic Cognitive Load** of both can be managed without requiring learners to make too strong a connection between them.

While a basic connection is necessary, too strong a connection would be undesirable because it would mean learners could only apply both subjects together and would have to completely relearn Object-Oriented Programming with a different language.

Finally, Cognitive Load can also be observed when understanding unfamiliar code. If the programming language and the requirement implemented in the code are known, the remaining **Intrinsic Cognitive Load** is lower than if one or both are unknown.

Additionally, the way the code is structured creates an **Extraneous Cognitive Load** that affects comprehensibility.
While formatting can be automatically adjusted, elements such as naming and comments can either increase or reduce. The same applies to the use of design patterns.

Cognitive Load in Teams

The Cognitive Load Theory has been developed over four decades and has been repeatedly validated for individual learning.

With the increasingly complex tasks related to digitalization and interdisciplinary problems, the question arises to what extent it is also valid and applicable for teams. The focus was not on how working in a team affects the Cognitive Load of individuals, but whether the concept of Cognitive Load Theory can be applied to a team.

The work of Paul Kirschner and colleagues demonstrated that this is fundamentally possible. However, unlike individual learning, there are additional aspects to consider and necessary adaptations.

The foundation is the hypothesis of a collective working memory within a team that exists only within that team. This collective working memory is formed through communication and coordination among team members. Each person contributes knowledge and helps connect shared understanding. This means new team members cannot automatically access what the team has already learned. The collective working memory thus provides a strong rationale for stable teams.

What's particularly compelling about the idea of collective working memory is that not every team member needs to learn everything, but individuals can focus on certain aspects. This makes it possible to manage the **Intrinsic Cognitive Load** of a very complex task by distributing it within the team. However, this doesn't work for every type of problem: While certain aspects of learning a business domain don't need to be understood by all team members, concepts related to tools are not easily divisible.

It may work effectively for a team implementing payroll accounting to have two people understand payroll tax and two others understand electronic reporting to health insurance companies. Such a division won't work with fundamental concepts: If two team members know how to declare variables in the programming language used, and two others know how to work with functions, no one masters the programming language well enough to use it effectively.

Kirschner and colleagues have shown that collective learning in a team works particularly well when the subject is too complex for an individual — meaning the **Intrinsic Cognitive Load** is too large — and the learning subject is equally new to all team members.

If the team has mixed levels of experience with the learning subject, negative learning outcomes can occur, particularly for team members who already have experience with the subject. These can be explained partly by group biases but are practically unavoidable.

An aspect that plays no role in individual learning has a dramatic effect on group learning: Transactive Activities contribute significantly to **Extraneous Cognitive Load**. Transactive Activities encompass all communication and coordination efforts that team members must make to learn together.
The costs of Transactive Activities can be so high that they negate the advantages of collective learning. It's therefore worth considering how these Transactive Activities can be reduced to minimize **Extraneous Cognitive Load**.

Reducing Cognitive Load

Sweller and colleagues have repeatedly demonstrated that high Cognitive Load hinders learning.
Reducing Cognitive Load has been a central goal in educational practice for many years.

For individual psychological **Cognitive Load Theory,** five principles have been identified that lead to a reduction of Germane Cognitive Load, with **Extraneous Cognitive Load** being the focus:

Coherence: simple, clear instructions and the elimination of superfluous images, sounds, or other stimuli.
Signaling: marking and highlighting the aspect that should be in focus.
Redundancy: non-competing elements with consistent content support learning, e.g., matching audio explanations for an image or labeled elements in a diagram.
Spatial contiguity: related elements are placed close together.
Temporal contiguity: related elements are visible simultaneously.

In addition, there are the didactic methods mentioned earlier to reduce **Intrinsic Cognitive Load** by meaningfully dividing the subject matter.

These methods naturally also apply to Cognitive Load within teams. However, as described above, additional factors can help reduce Cognitive Load in teams:

Complexity of the learning subject: The task must be complex enough to justify the additional effort of collective learning for all team members.
Guidance and support: Supporting and guiding the team during learning through suitable environment, resources, and methods reduces Transactive Activities.
Domain experience: The more experience team members have in the domain, the fewer Transactive Activities are required.
Collaboration experience: The more experience team members have working together, the fewer Transactive Activities are necessary.
Team size: Smaller teams require fewer Transactive Activities.

Roles: Clearer roles for team members in the learning situation reduce Transactive Activities.
Team composition: More heterogeneous teams require more Transactive Activities.
Work experience: More experienced team members require fewer Transactive Activities with the learning subject.
Previous collaboration: Team members with more experience working together need fewer Transactive Activities. Transactive Activities represent the main driver of **Extraneous Cognitive Load** when learning in teams.

Achieving More Together

The **Cognitive Load Theory** effectively explains which factors support or hinder learning. For individuals, it has been well researched since the early 1980s and is empirically validated.
The application to groups has been specifically studied for about 15 years, and it appears that the basic principles of **Cognitive Load Theory** also apply to team learning.
These insights form the foundation for numerous tools and methods developed and applied in software engineering in recent decades, such as workshop formats like Event Storming, structuring methods for requirements like User Story Mapping, or the development of a common language between business departments and software engineering.
One thing is clear: The number of tasks that exceed a single person's cognitive capacity is increasing and will continue to grow. However, with insights from **Cognitive Load Theory**, we can find ways to learn together and successfully tackle these challenges. And that's an encouraging thought.

Gerrit Beine
INNOQ

Internal Development Platforms – Shift Down Instead of Shift Left

Internal Developer Platforms (IDPs) help us accelerate software development. They represent the natural evolution of developments over the past decades. We'd like to show why this is the case with a historical overview and then examine current challenges.

The Lowest Imaginable Autonomy

Those of us who have been around remember: Before the era of agile software development, we had large phase models like the waterfall model. Users would wait a long time before seeing any running software. Handoffs between teams, such as between development and QA, were complex and time-consuming. None of the teams involved had true autonomy. Each team within the development process was heavily dependent on at least the teams immediately before and after it. However, the mental load on developers was comparatively low. Developers primarily needed to know design principles, programming languages, and frameworks.

Agility Increases Autonomy, But Also Mental Load

Agility emerged in the late 1990s with models such as Extreme Programming, Scrum, and Crystal. At the time, these approaches promised rapid software delivery in 6-week cycles compared to the usual 6 to 24 months. This was partly because many handoffs were eliminated. The development team became "cross-functional," incorporating different roles such as requirements engineering, development, and testing into a single team. However, security and operations still remained separate from the development team. The mental load increased significantly. Beyond software development, developers now had to write automated tests, often speak directly with users, estimate user stories, present results, and much more.

Shift Left, DevOps, and "You Build It, You Run It"

After the publication of the Continuous Delivery book (2010)[5] and Werner Vogels' famous interview about software development at Amazon (2006)[6], operations also moved into the development team's realm ("You Build It, You Run It"). The term "Shift Left" became popular.

The term originated from the waterfall model's visual representation from left to right. "Shift Left" means moving tasks such as testing or operations "to the left" into earlier phases of the process, such as architecture or development.

Development teams that successfully shifted many activities to the left could deliver high-quality software very quickly. Why? They had very high autonomy. Speed comes from shortening or even eliminating handoffs between teams.

However, the mental load for these teams increased considerably. Operations and infrastructure were added to their existing responsibilities, including maintaining CI/CD pipelines, managing cloud technology, participating in on-call rotations, and implementing observability.

Mental Overload: From Shift Left to Pile Left

With the introduction of concepts like DevSecOps, FinDev, and FinOps, development teams were expected to have more and more capabilities. Many companies found this was no longer manageable. It was no longer "Shift Left" but "Pile Left."

Development teams suddenly had too many responsibilities. In the name of high autonomy and delivery speed, they were supposed to handle everything. In some cases, development teams were even expected to manage license purchases.

"Pile Left" inevitably led to problems. There are limits to what a development team can and should do. UX, security, QA, infrastructure, operations, and other areas are complex disciplines requiring specialized skills. Very experienced teams with a good mix of competencies might manage these tasks, but companies with multiple teams cannot expect all teams to be set up accordingly or to learn the necessary skills quickly enough. This approach isn't efficient anyway.

The complexity of developers' lives has steadily increased over the past 25 years. What was unthinkable in the late 1990s during the emergence of agility — developers being responsible for test automation — is completely normal today. Additionally, we now work with larger and more complex software systems requiring more developers across more teams. Both factors make it worthwhile for many companies to invest in Enabling Teams and Internal Developer Platforms. These help better organize redundant work and knowledge building, thus managing the increasing complexity.

With Platforms and Enabling Toward Shift Down

If high autonomy creates a high mental load for many development teams, solutions must be found. Currently, there are two complementary approaches: Internal Developer Platforms (IDPs) and Enabling Teams[7].

IDPs try to "hide" as much of the specialized knowledge mentioned above in the platform as necessary ("Shift Down"). For example, they can handle infrastructure provisioning, monitoring, dashboards, log infrastructure, CI/CD, security (e.g., secrets management, vulnerability checks, authorization and authentication), license scanning, as well as application and service templates. This means developers no longer have to worry about "everything," as the IDP handles the relevant work in the respective context.

Caution is needed here: Too much functionality in an IDP can drive up costs. Additionally, developers may be deprived of the necessary freedom to innovate on the platform.

IDPs can and should also address domain-specific infrastructure issues. An IDP could, for example, provide a "customer database" that meets operational requirements for locality, encryption, and data backup for customer data. This goes beyond providing a simple cloud database that would still need to be configured accordingly (Hohpe and Seitz[8]).

A platform can do a lot, but it cannot remove all mental load related to infrastructure, security, or testing from development teams. For these gaps, "enabling" is needed — coaching and support from specialized teams. These teams bring together experts with the task of empowering other teams to master specific challenges.

Success Factors for Internal Developer Platforms

Building and operating a successful IDP is often underestimated. An IDP is an internal product with its own customers (developers, operations, business) and therefore requires everything a product needs: besides a development team, it

needs product management, good requirements engineering, support, an excellent developer experience, self-service capability, and internal marketing.
Since this requires considerable effort, an IDP is usually only worthwhile from a certain size of development organization. The leaner the IDP, the sooner a company can afford one.

Looking Beyond: Domain-Specific Platforms

IDPs are not the only internal platforms that exist in companies. There are also internal domain-specific platforms, sometimes called "common services." These offer functions that cover frequent and recurring requirements of many development teams in the corresponding domain, such as payment processing, fraud detection, or data analysis. These domain-specific platforms must be identified as important business components, developed, made known internally, and made discoverable. As with IDPs, the same rule applies: They are internal products and require the same organizational prerequisites and efforts to be successful.

Conclusion

A cross-functional development team can deliver software significantly faster than a traditional development team because ideally, it has no handoffs with other teams and therefore doesn't have to wait for external backlogs to be processed. However, it must take on a variety of different tasks. This costs time and resources until the necessary skills are learned, and leads to a very high mental load when team members switch between these tasks.

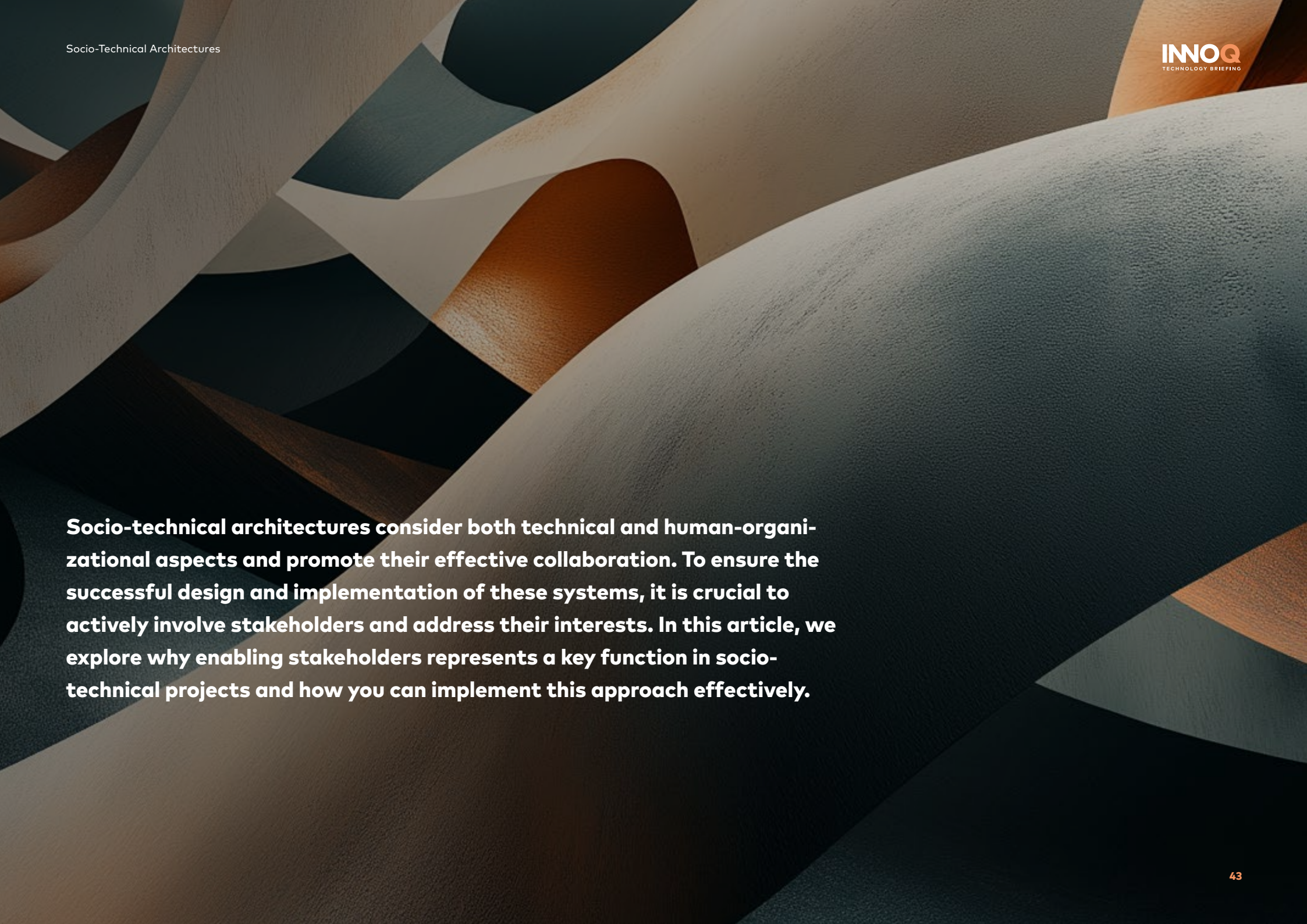
An IDP (Internal Developer Platform) — an internal company product with product management, support, internal marketing, and self-service capability — can significantly reduce these challenges.

It reduces both the "learning costs" that must be paid and the mental load of team members. The remaining gaps can be further addressed with Enabling Teams.

Sven Johann
INNOQ

[5] Continuous Delivery, Jez Humble und Dave Farley, 2010, https://www.oreilly.com/library/view/continuous-delivery-reliable/9780321670250/
[6] ACM Queue, Interview with Werner Vogels (CTO amazon.com), 2006, https://queue.acm.org/detail.cfm?id=1142065
[7] Team Topologies, Matthew Skelton und Manuel Pais, 2019, https://teamtopologies.com/book
[8] Entwickler skalieren anders als Applikationen, 2024, Gregor Hohpe und Johannes Seitz, https://www.innoq.com/de/articles/2024/09/entwickler-skalieren-anders-als-applikationen/

Enabling Stakeholders as a Success Factor



Socio-technical architectures consider both technical and human-organizational aspects and promote their effective collaboration. To ensure the successful design and implementation of these systems, it is crucial to actively involve stakeholders and address their interests. In this article, we explore why enabling stakeholders represents a key function in socio-technical projects and how you can implement this approach effectively.

What Does Enabling Stakeholders Mean?

"Enabling stakeholders" means empowering and supporting the involved interest groups so they can actively and competently participate in a change process. This specifically includes:

- imparting knowledge needed to make informed decisions,
- providing relevant information, and
- promoting a common understanding as well as a clear communication culture.

The goal is for all participants to understand and effectively consider the overarching objectives, the technical and social aspects of a solution, as well as the business requirements across departments.

Why Is Enabling Stakeholders Crucial?

Mastering Interdisciplinary Complexity

Socio-technical architectures are characterized by close integration of technical and human factors. They require stakeholders to understand and manage both the technical systems and business processes.

An isolated approach that focuses only on technology or the interests of individual stakeholders typically leads to incomplete or poorly fitting solutions. However, when management and teams are enabled to equally consider technical and cross-departmental business aspects, a foundation for sustainable adaptability emerges that extends beyond departmental boundaries.

Promoting Acceptance and Motivation

An architecture can only succeed if the people involved consider it valuable and actively contribute to it. When stakeholders understand the goals and benefits of the system, they are more motivated to support its implementation. Without this acceptance, projects risk facing resistance or being implemented half-heartedly, resulting in inefficient or inappropriate solutions.

Faster Adaptability in a Dynamic Environment

Our business world is characterized by constant change. Markets and technologies evolve rapidly, and companies must respond flexibly. This agility can only be ensured if stakeholders have the necessary competencies and knowledge to continuously adapt and develop the system. Empowered leaders and competent teams are invaluable in this context.

How Can We Successfully Enable Stakeholders?

To systematically and purposefully design stakeholder enabling, the following approaches have proven effective in practice:

Stakeholder Analysis and Communication Strategy

At the outset of a project, conducting a stakeholder analysis is essential to identify the needs, expectations, and potential resistance of various interest groups. This informs a communication strategy that brings all participants to a common level of understanding and actively involves them in the process. Transparent communication fosters willingness to engage with new solutions and creates the necessary foundation of trust.

Knowledge Transfer

Building shared knowledge is central to enabling stakeholders. This includes ensuring that management and departments develop a fundamental understanding of technical aspects and business processes – even across departmental boundaries. Through knowledge-sharing sessions, decision-makers can learn both the technical and business fundamentals necessary for cross-departmental decision-making.

Iterative, Collaborative Development Process

A successful socio-technical system doesn't emerge overnight but through an iterative, adaptable development process. An agile approach with regular feedback and adjustments ensures that the system continuously aligns with stakeholder needs. These iterations also foster collaboration and ensure the system can flexibly respond to changes.

Visualization Tools

Tools that document and visualize both technical and business process aspects of the architecture help make complexity more tangible. They provide a common basis for discussions and facilitate coordination between different departments. This transparency supports stakeholders in their decision-making and promotes cross-departmental understanding.

Who Takes Responsibility for Enabling Stakeholders?

The responsibility for enabling stakeholders often falls to specialized roles or teams that combine both technical and organizational expertise.

A prominent model suited for this purpose is the Enabling Teams described in Team Topologies. These teams act as supporters and facilitators who help other teams achieve their goals more effectively through targeted interventions.

An Enabling Team works closely with involved departments, identifies their specific needs and challenges, and provides tailored support – whether through training, workshops, or developing customized tools and processes.

When developing and implementing new collaboration or decision-making models for the entire organization or specific divisions, building a dedicated team makes sense. This team identifies appropriate approaches and communicates their application to all involved parties.

If the procedures are already clear to all participants and it's primarily about mediating cross-departmental interests and their technical feasibility, individual experts can also share their knowledge and facilitate joint decision-making.

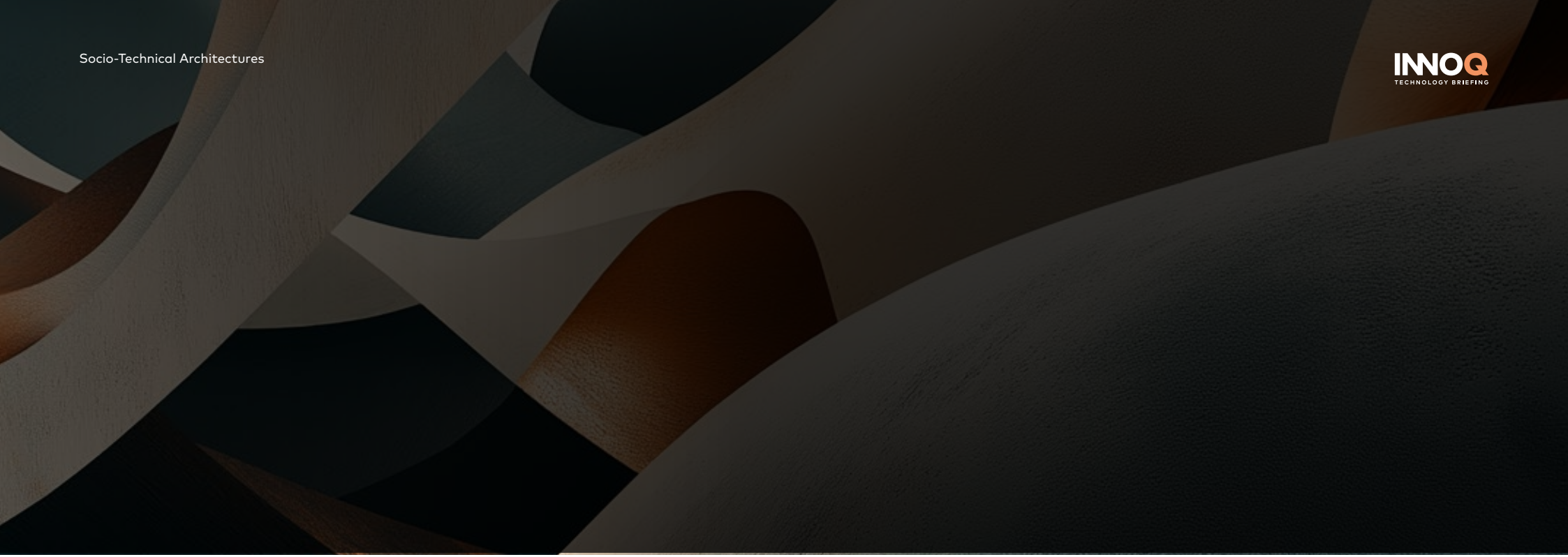
Conclusion

Implementing socio-technical architectures is virtually impossible without active participation and empowerment of stakeholders. IT decision-makers who invest specifically in enabling management and teams not only create the foundation for the acceptance and sustainability of these systems but also foster a culture of collaboration and adaptability.

Clear communication, effective knowledge transfer, and building a learning organization are central success factors for remaining competitive in an increasingly dynamic and interconnected world.

André Aulich
INNOQ

Socio-Technical Architectures: Informality from Mining to Today[9]



To understand the concept of "socio-technical systems," it's worth looking back to the beginning of the 20th century.

At that time, the "economic man" perspective dominated, where people were primarily motivated by financial incentives. The focus was therefore on payment, bonuses, and piecework. About 20 to 30 years later, this view shifted toward the "social man." Now, social relationships and leadership styles were considered key to promoting motivation and satisfaction.[10]

During this period, coal was being mined extensively in Great Britain. Coal miners worked predominantly in autonomous groups that organized their own work processes. Despite established safety procedures, they enjoyed considerable freedom. From the 1940s onward, however, working conditions changed due to mechanization and specialization.

Work areas expanded, making communication more difficult, and the decision-making freedom of these groups was severely restricted.[11]

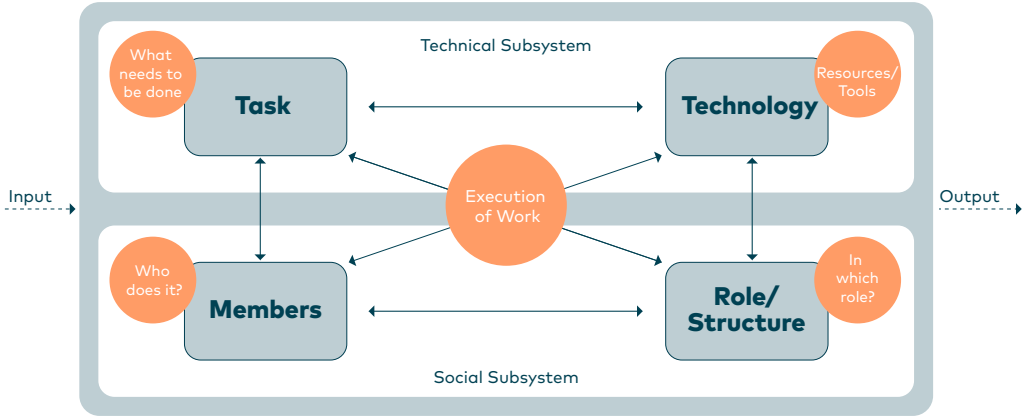
The result was a breakdown of social bonds and increasing conflicts. Both productivity and job satisfaction declined.

In the 1950s, research by Trist and Bamforth[12] showed that increasing bureaucratization and the loss of informal processes negatively affected efficiency. The concept of "socio-technical systems" emerged from this research.

What is a Socio-Technical System?

A socio-technical system is a complex work system in which technical and social subsystems jointly perform tasks.

The **technical subsystem** encompasses the tasks and the supporting or executing technology. The **social subsystem** describes the members, their roles, and the structure in which they operate. The technical system creates framework conditions for the social one, while the social system drives the evolution of the technical system.[13]



What Does This Have to Do with Us?

Why is INNOQ interested in the history of mining? Because the parallels to today's organizational structures can teach us a lot. In the past, miners worked in fixed systems – today we enter our clients' often rigid socio-technical systems as external professionals.

Like the miners back then, we too must adhere to informal rules and unwritten laws that can strongly influence project dynamics. An example from our practice shows how such informal structures become visible in development projects and how they shape our approach.

Case Study: The internal Organization App

INNOQ was commissioned to develop an internal app for a company where working conditions have changed significantly due to the Covid pandemic. The app is intended to serve as a flexible solution to adapt work organization and internal communication channels to the new reality.

The following areas are affected and should be addressed in the app:

Office Spaces

Many colleagues work at least part-time from home or even in other countries. To accommodate this situation, the company has already downsized its office space. In fact, there are no longer enough workstations for everyone. Management therefore wants a system where desks can be booked.

Cafeteria

Similar challenges apply to the new cafeteria, which is also not designed for a large number of people. However, there are "peak periods" when the number of tables is insufficient.

Communication

In addition, internal interaction and exchange have become much more asynchronous, as employees no longer automatically work in the same location (and often not in the same time zone).

The initiative for the app came from management, not from the staff, which already creates potential hurdles.

Informal Structures

This shift in everyday work brings informal structures into play that the company has not officially established but which are nonetheless firmly anchored in daily routines. Such structures are not conscious directives from management or the works council but emerge through established practice. In this context, Stefan Kühl refers to "undecided decision premises"[14] that have been established through regular practice.

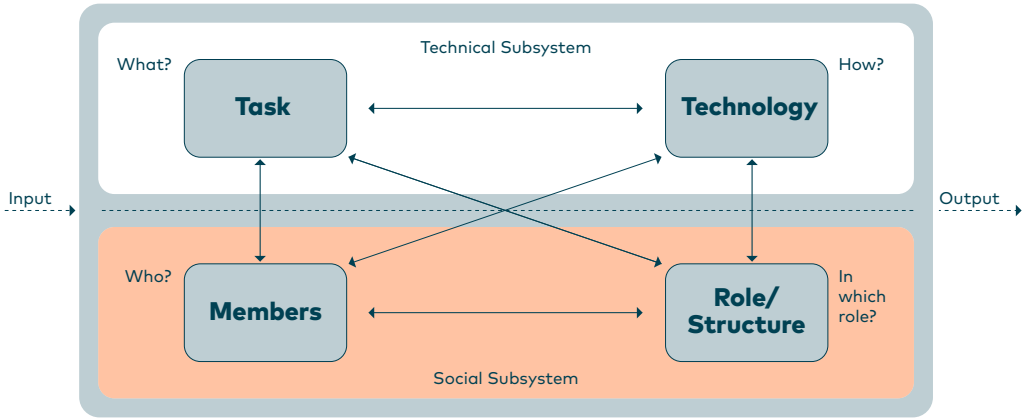
This isn't a one-time improvisation – these structures are more like a well-trodden path. However, no documented decisions about this path can be found anywhere. Informal structures can be very stable and long-lasting, even persisting when regulations exist that contradict them.[15]

In practice, organizations frequently toggle between formality and informality. In fact, organizations benefit from informality. We all know the expression "work-to-rule," which simply means adhering exclusively to formal rules. However, this expression is often understood almost as a form of protest.

Informal structures are therefore by no means exclusively negative for an organization. They are adaptable (unlike technology), promote innovation, and contribute significantly to the company culture.

What Does This Mean for the App INNOQ is Supposed to Develop?

As a reminder: Informal structures are primarily found in the lower part, in the social subsystem.



The **members** of the system are the users of the app – i.e., the employees and management of the company – but also us as external developers. These members operate in formal **structures and roles**, such as departments or project management, which are often clearly documented.

Alongside these, informal structures exist that only long-term members know and which vary greatly depending on the area. The **task** set by management is to support internal work organization and communication through the app – even though management itself may not have a complete picture of all informal processes.

Technologically, the app should flexibly support users in their way of working without creating unnecessary complexity. It needs to function without dictating processes too rigidly. How the software allows users to intuitively control work processes and accommodate informal, established procedures plays a crucial role here.[16]

Challenges in Developing the App

The application formalizes and bureaucratizes processes that previously ran informally (e.g., finding a free desk in the office or communication channels). Recall the mining example: There too, processes were formalized and established when machines came into play.

If the app fails to reflect informal structures, it may face resistance. It will then only be used to the extent absolutely necessary, while informal structures will likely continue to exist in parallel.

If the application ends up not being used, the project has failed. However, it can be extremely difficult to formalize shadow processes – as they are often called in the literature. This can become an extensive and complicated undertaking, as informal structures are highly dynamic and adaptable, which is often not the case with technology.[17]

Informal processes are often invisible because they are not documented, are based on relationships, and are influenced by subtle signals.[18]

What to Do?

How can we, as newcomers, recognize the informal structures so that we can take them into account when developing the app?

1. Requirements Analysis:
Informal processes are dynamic and based on human interactions. To capture them, we speak not only with management but especially with the end users.
Methods such as sociomatrix, sociogram, or collaborative modeling help to map relationship networks and bring together expertise.

2. Build Systems to be Expandable and Flexible:
The app must be designed flexibly and leave room for adaptation. Development should be user-centered and agile, with regular user tests and feedback.
It is particularly helpful to observe users in real scenarios, even with prototypes. Expandability and flexible use are crucial – even if this makes the UX more complex. Evolutionary architecture, lean methods, and agile development offer valuable approaches here.

3. User Testing:
Feedback is worth its weight in gold. Prototypes and MVPs (Minimum Viable Products) should be tested early – long before the final implementation. This allows us to directly check whether the application truly meets the requirements and will be well received. MVPs help to develop the most important functions in a targeted way from the beginning, without needing to complete the entire product first.

Conclusion

In the context of socio-technical systems, it becomes apparent that the interplay of social and technical structures is crucial. The challenges faced by miners with new technology are also found in modern work environments today: humans and technology are in constant interaction and shape each other.

Lena Kraaz
INNOQ

[9] Based on the podcast by Katharina Baur and Lena Kraaz entitled "Soziotechnische Systeme – Informalität vom Bergbau bis heute", recorded on 08.04.2024
[10] Prof. Dr. Simone Kauffeld, Dr. Nils Christian Sauer (2019): Vergangenheit und Zukunft der Arbeits- und Organisationspsychologie. Springer Berlin Heidelberg.
[11] Trist, E. L., & Bamforth, K. W. (1951). Some social and psychological consequences of the Longwall method of coal-getting. Human Relations, 4(1)
[12] Tom Galvin, Pedro Monteiro, Miranda Lewis, Joe Bradley (2017): Sociotechnical Systems – Trist and Bamforth. Talking about Organizations Podcast, episode 34 https://www.talkingaboutorganizations.com/e34/
[13] Page "Soziotechnisches System". In: Wikipedia – The free Encyclopedia. State: 08.11.2024. URL: https://de.wikipedia.org/wiki/Soziotechnisches_System
[14] Kühl, Stefan (2010): Informalität und Organisationskultur - Ein Systematisierungsversuch. Working Paper 3/2010
[15] Tretschok, Katja (2019): Die Relevanz von informellen Strukturen innerhalb von Organisationen Masterarbeit, Hochschule Mittweida, Fakultät Soziale Arbeit.
[16] Page "Soziotechnisches System". In: Wikipedia – The free Encyclopedia. State: 08.11.2024. URL: https://de.wikipedia.org/wiki/Soziotechnisches_System
[17] Sutter, Anke et al (2021): Soziotechnische Systeme : Der Mensch in der Industrie 4.0
[18] Kühl, Stefan (2010): Informalität und Organisationskultur - Ein Systematisierungsversuch. Working Paper 3/2010

INNOQ

Our consultants have been advising SMEs & corporations for over 20 years, implementing IT systems of all sizes.

Our expertise is drawn from extensive hands-on experience in software architecture and development, platform operation and infrastructure, as well as digital product development.

We don't view **technology as an end in itself,**
but as an **enabler for solving real-world problems.**

www.innoq.com
info@innoq.com
Tel. +49 2173 33 66 0

You don't want to miss an issue of the INNOQ Technology Briefing?
You can subscribe here: **briefing.innoq.com**

We assist you

Talk to us about your plans, even if they're just rough, initial thoughts. We'll support you specifically or throughout the entire modernization journey.

Strategic support: What's the best strategy for you? What makes the most sense in relation to your business goals? Together, we define initial milestones.

Development: Complete or partial implementation of your IT modernization project by one or more teams of interdisciplinary professionals.

Team: A team of experts will support you, taking the lead or integrating into your team structure, for strategic, architectural, and technical tasks in your modernization effort.

Consulting and workshops: We will co-develop a strategy for your company's IT modernization. With workshops like Quality Storming and Big Picture Event Storming, we efficiently explore potentials.

Reviews: Our experts will take an in-depth look at your organization's legacy systems. You'll get valuable insights and an honest external assessment. Along with clear recommendations for action.