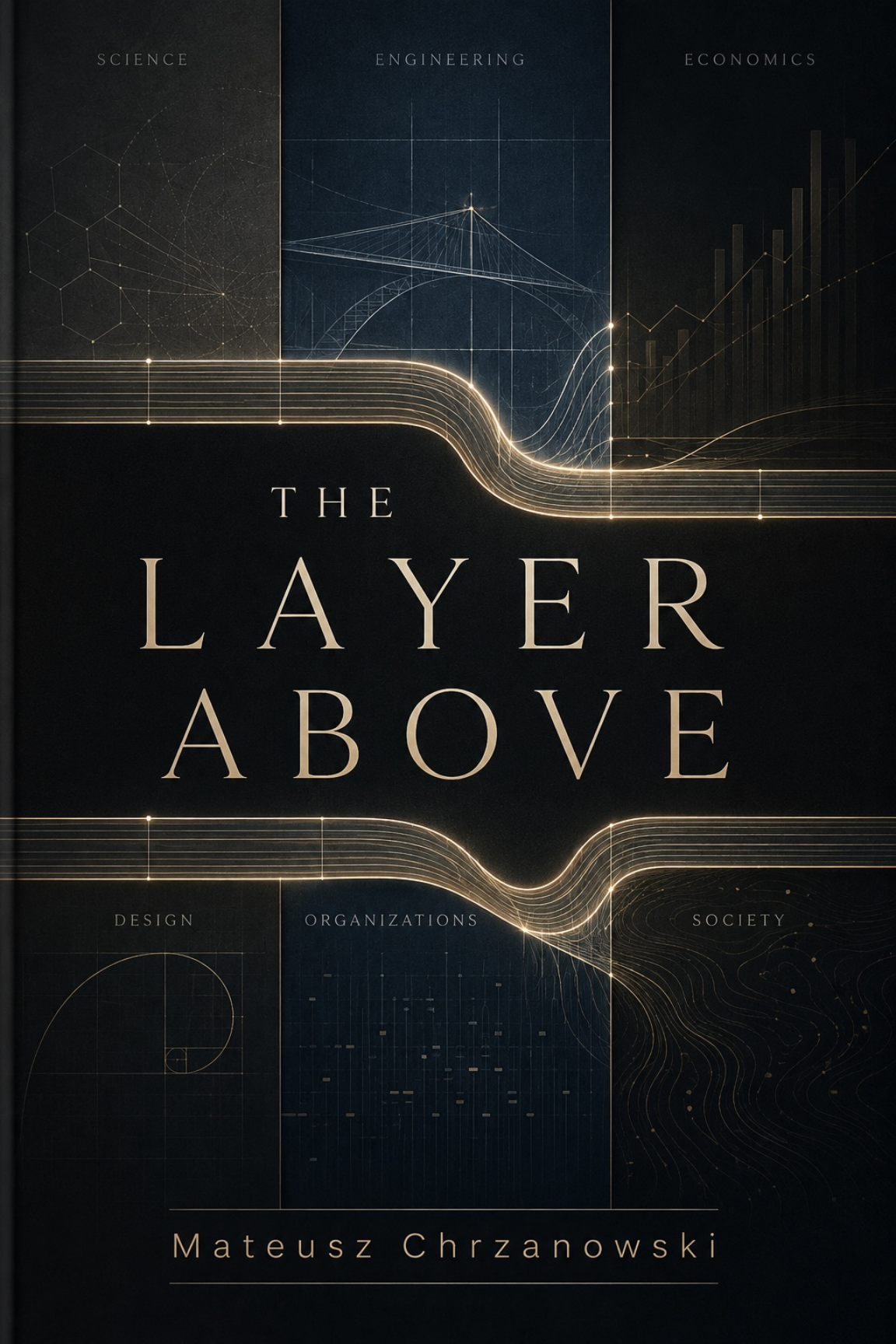


Mateusz Chrzanowski

THE LAYER ABOVE

Mateusz Chrzanowski

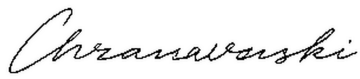
Archestral Studio

© 2026 Mateusz Chrzanowski

Published by Archestral Studio
First Edition 2026

Paperback ISBN: 978-83-982835-0-2

I do not intend to teach you anything in this book.
I intend to show you a different perspective, and I hope it stays with you.



The perception of the object depends on the direction of the eye.

— Leonardo da Vinci

Contents

Prologue 1
Two Systems. One Function.
Chapter 1 13
The Problem Started Before You Saw It
Chapter 2 29
Map Reality, Not the Organisation Chart
Chapter 3 54
The Boundary Is Not the System
Chapter 4 81
The System Stops Being So Surprising
Interlude I 97
Pandemonium: Rebuild a Live System
Chapter 5 104
Architecture Accumulates. So Does the Debt.
Chapter 6 119
What the Dashboard Cannot See
Chapter 7 147
Optimised Fragility
Chapter 8 161
Start with the Function, Not the Tool
Chapter 9 186
When the Domain Stops Being a Boundary
Chapter 10 203
An Idea Is Not an Architecture
Interlude II 229
JapoJade.pl: Create a Missing System
Chapter 11 243
The Product Is Not the System
Chapter 12 258
The Company Is Not the Boundary
Chapter 13 279
Responsibility Without Authority Is Theatre
Interlude III 289
The Power Plant: Make Complexity Executable
Chapter 14 302
System of Systems Architect
Epilogue 320
The Layer Above

Prologue

Two Systems. One Function.

A platform for truck drivers and the construction of an industrial power plant should have almost nothing in common. One belongs to transport, labour, technology and business. The other belongs to energy, automation, electrical infrastructure and industrial construction. One is built around drivers and companies trying to find each other in a fragmented market. The other is built around control cabinets, cables, automation devices, HVAC systems, commissioning and thousands of physical connections that eventually have to work together.

By normal professional standards, these are two different worlds.

I entered both.

Japojade.pl began with a market that already contained everything required for transactions to happen, but lacked the system that could organise them. Drivers had availability. Transport companies had demand. Routes existed. Rates existed. Jobs existed. People were already moving between companies through informal networks, phone calls, recommendations and fragmented online channels. The market was alive, but structurally incomplete.

The obvious response would have been to build another recruitment website.

That was not the problem I saw.

The problem was that the market lacked an organising layer. Information about supply and demand existed, but it was scattered. The structure required to connect driver availability with company demand, route, timing and commercial conditions did not. So I

© 2026 Mateusz Chrzanowski · Archestral Studio

designed one. Japojade eventually attracted more than 6,000 driver-side participants and approximately 1,000 company-side participants.

Years later I was working inside a completely different system. Two gas-fired power units. Sixty-three control-and-power cabinets. Approximately eleven kilometres of cable routes. More than one hundred kilometres of power and control cabling. Around thirteen kilometres of protective tubing for fibre optics. Dozens of HVAC and heating devices, building automation and a central control environment.

This time the system already existed.

At least technically.

The equipment had been designed. The documentation existed. The final installation had a defined purpose. What did not yet exist was a structure capable of turning that technical complexity into executable work. The available workers had not arrived as an experienced industrial automation team. Yet cables still had to be routed, cabinets connected, automation devices installed, measurements controlled, documentation maintained and work coordinated with commissioning engineers. A technically correct project does not solve any of those problems by itself. It defines the destination. Someone still has to create the path from drawings to physical reality.

So the work was decomposed until people could execute it. Technical complexity became sequences, instructions, connection logic, material flows and verification. Workers who had not previously operated as industrial automation specialists became capable of performing parts of that work correctly because the system around their work was redesigned.

Japojade required a system to be created where the market had only fragments.

The power plant required complexity to be converted into execution.

Different domain. Different material. Same function.

That is the point where the usual way of describing professional competence starts to fail.

© 2026 Mateusz Chrzanowski · Archestral Studio

Most careers are organised around domains for a very good reason. Society needs doctors, electricians, lawyers, software engineers, production managers and thousands of other specialists. The classification works because it works for most people. A person enters a field, acquires knowledge, gains experience and becomes increasingly competent inside its boundaries.

There is nothing wrong with that model.

It simply does not describe everyone.

Some people operate differently. Their primary competence is not attached permanently to one domain. They enter a system, learn what the system requires, reconstruct how its parts interact and operate on the architecture above the individual specialisations. That difference sounds abstract until you stop looking at industries and start looking at systems. A company is a system. A factory is a system. A market is a system. A government agency is a system. A city is a system. A continent is a system. A relationship between two people is also a system. So is the human body. So is the human mind.

They are obviously not the same systems. Their components, scale and physical reality can be radically different. But all of them have an architecture. And architecture exists whether anyone designed it or not. A new company can begin with ten employees and a founder. At that scale, the architecture may actually be deliberate. Everyone knows who sells, who buys, who makes decisions, who controls quality and who speaks to the customer. Information moves through a small number of people, and the founder can still see most of the company directly.

Then the company succeeds.

Ten people become fifty. Fifty become two hundred. Production moves into its own facility. Sales expands. Management layers appear. New systems are purchased. Someone creates a procedure to solve a recurring problem. Another person creates a spreadsheet because the main system does not provide the information needed to do the job. A department improves its own result and quietly transfers part of the cost somewhere else. New managers reorganise pieces of the company. Temporary solutions remain because replacing them would interrupt operations. New technology is connected

© 2026 Mateusz Chrzanowski · Archestral Studio

to old technology. Reporting expands because direct visibility disappears.

Eventually eight hundred people may be working inside an organisation whose original architecture was designed for ten.

The company still has an architecture.

Most of it was simply never designed.

It accumulated.

This distinction is fundamental. We often speak about badly designed systems as if every part of them must have been deliberately created by someone. In reality, much of what we call architecture is the residue of thousands of decisions made at different times, by different people, for different reasons. Each decision may even have made sense locally. The resulting whole may still make no sense at all.

I do not begin by assuming that the current structure deserves to exist.

I begin with why it exists.

That is different from asking how to improve it. Improvement starts from the existing system and tries to make it better. I am interested first in whether the existing structure should survive at all. The question is simple: why does this work this way? Then I keep going until I reach the layer that exists because reality requires it rather than because someone, somewhere, once decided that this was how things should be done.

A process may exist because twenty years ago the technology required it. A reporting structure may exist because management once lacked access to information that is now available instantly. A physical operation may still be organised around limitations of equipment that has already been replaced. A regulation may force a specific constraint. A habit may survive without any constraint behind it at all.

Those are not equivalent.

If the distinction is not made, history becomes confused with necessity.

Once the accumulated structure is stripped away, the problem becomes much cleaner. A system has a function. It operates inside

© 2026 Mateusz Chrzanowski · Archestral Studio

physical reality. It must achieve something. From that point it can be rebuilt using the knowledge, technology and capabilities available now, not the assumptions inherited from the system that came before it.

And there is one boundary I treat differently from all the others.

Physics.

If physics makes something impossible, it is impossible.

If physics allows it, everything after that is an execution problem.

That distinction matters because organisations constantly describe secondary constraints as if they were fundamental ones. They do not have the right tool. They do not have people with the right competence. Nobody in the industry works that way. The available technology is normally used for something else. The current organisation cannot support it.

None of those statements establish impossibility. If a tool does not exist but can physically be created, the tool can be designed. If people do not have the required competence, the work can be reorganised, competence can be built or people with different competence can be brought in. Capital, resources and time decide how an architecture reaches reality. They do not change the boundary of what reality permits.

This is why I separate architecture from vision. A visionary can describe a world that current physics or technology does not yet allow us to build. Researchers may then spend years moving the scientific boundary. Engineers may spend another decade converting that knowledge into usable technology. The vision can be valuable long before it becomes executable.

That is not the work I am describing here. A System of Systems Architect works with reality that can be engineered. Known physics. Existing technology. Technology close enough to existence that it can be incorporated into a real development path. If a missing component can be created within those limits, its absence is a design problem, not a wall.

The same principle destroys another artificial wall: the boundary between industries.

Technology does not care which industry invented it.

© 2026 Mateusz Chrzanowski · Archestral Studio

A mechanism used in a truck does not become structurally unavailable to automotive design because the vehicle is smaller. A control method developed in aviation is not owned by aviation. A logistics principle does not stop functioning when moved into medicine. A technology does not know whether a professional classification system has placed its original application under transport, manufacturing, healthcare or consumer products.

It performs a function.

Functions can move.

I once applied this way of thinking to something as ordinary as a car seat. Heavy trucks and some vans use pneumatic seat suspension to isolate the driver from movement and vibration. The existing implementations are much larger because they work inside a different physical environment. But the function itself is simple: separate the seat from part of the vibration transmitted through the vehicle.

There is no reason for that function to belong only to a truck. Shrink the principle. Use a very small pneumatic layer controlled by compact compressors. When the car starts, the seat rises by only a small amount – barely noticeable to the driver – but it is no longer resting on the body of the vehicle in exactly the same way. The suspension deals with the movement of the car. The seat deals with the smaller vibration still travelling through the body.

It is a small change.

After ten minutes, it may mean almost nothing.

After eight hours, small changes stop being small.

That idea did not require inventing a new law of physics. It required ignoring the industry boundary around an existing function. This is what cross-domain work actually means. It does not mean knowing everything about every industry. Nobody does. It means that the domain does not define the edge of the architecture. When a system requires knowledge from a new field, that knowledge becomes another part of the model to acquire.

This is also where the difference between knowledge and physical skill becomes important. I can learn how an electrical system works and design one. I can study medicine deeply enough to understand biological mechanisms, pharmacology, diagnostic logic and

© 2026 Mateusz Chrzanowski · Archestral Studio

the relationships between conditions. I can analyse legal structures and understand how rules interact. I can enter transport, industrial automation, automotive design or another technical field and build a model of what is happening there.

That does not mean I can automatically perform every physical skill inside those domains. Understanding surgery does not make someone a surgeon. Knowing exactly what a violinist is doing does not give the left hand thousands of hours of trained movement. Physical mastery belongs to another category of competence. The distinction matters because professional categories usually combine the two. A domain becomes identified with the specialists who work inside it, and over time we start treating the boundary of their profession as the boundary of the system itself.

It is not.

The system is larger than the profession.

This becomes obvious as soon as a problem crosses enough interfaces.

A factory is not only production engineering. It is also people, logistics, maintenance, information, finance, suppliers, energy, quality, management, physical space and time. A hospital is not only medicine. A transport company is not only vehicles. An energy system is not only generation. A business is never merely the sum of the departments shown on its organisational chart.

Specialists are necessary because each layer can contain enormous depth. The error begins when depth inside one layer is mistaken for visibility of the whole. The higher you move, the less useful industry boundaries become. At the highest level, the material can change completely while the architectural logic remains recognisable. There are actors. There are resources. There are dependencies. Things flow. Information moves. Decisions are made somewhere. Control exists somewhere. One failure changes the state of something else. Local actions propagate through the system. Time changes the result. The system reacts.

© 2026 Mateusz Chrzanowski · Archestral Studio

That can describe an organisation employing eight hundred people, a power grid, a supply chain, a market, a small shop or a relationship between two people. The individual variables are different. The architecture still exists. That is why the jump from a driver labour market to industrial automation is much smaller to me than it looks from the outside.

In transport, I had to understand the variables that defined the market and build the missing layer between them. At the power plant, I had to understand enough of the technical system to divide complexity into executable work and keep multiple layers moving toward one physical result.

I did not need the two domains to resemble each other.

I needed to see the systems.

This is also why I do not describe the role as consulting.

Consulting has a different function.

A healthy company may want to understand how its position compares with competitors, how a market is changing, what solutions already exist or what a particular technology can offer. Those are legitimate consulting questions. The company itself may be functioning well. What it needs is information, comparison or specialist knowledge about a defined field.

An architect can use a consultant for exactly that reason: to bring specialised market, technology or domain knowledge into a larger architecture. Rebuilding the company itself is a different problem. So is rebuilding an entire department, reconstructing its operating logic or creating a new organisational structure from zero. That is not what consulting is for.

The object being changed is no longer a defined external field. It is the system itself. A department does not exist independently of the people, processes, information, technology, responsibility, costs, interfaces and other parts of the organisation around it. A company cannot be optimised as a collection of narrow functions and assumed to be optimised as a whole.

The perspective is too narrow for the boundary of the problem.

Once the system itself has to be reconstructed or rebuilt, the required function changes.

© 2026 Mateusz Chrzanowski · Archestral Studio

Architecture has to survive contact with execution.

Japojade could not remain a diagram of how a labour market should work. It had to become a functioning platform used by real drivers and real companies. The power plant could not remain a technically elegant set of drawings. Cables had to physically arrive at the right cabinets. People had to know what to connect. Measurements had to be completed. Interfaces with other teams had to work. Eventually the system had to move toward commissioning.

The output of architecture is not the elegance of the model.

The output is a system that can exist.

That is the common function behind the two stories at the beginning of this prologue. One involved creating a structure that the market did not have. The other involved taking a structure of enormous technical complexity and making it executable by real people under real conditions.

The material changed.

The function did not.

This book calls that function System of Systems Architecture.

The term matters because most serious systems do not exist alone. They contain systems and depend on other systems. A company contains financial, operational, human, technical and information systems at the same time. A large industrial project adds physical infrastructure, suppliers, logistics, quality, contractors, commissioning and external organisations. A national system may contain thousands of organisations that nobody controls from a single point.

Looking at one system is often not enough.

The result may be created by the relationship between several of them.

That relationship is easy to miss because organisations divide reality into manageable pieces. Departments have boundaries. Professions have boundaries. Companies have boundaries. Industries have boundaries. Governments have boundaries. Reality does not have to respect any of them. A decision made in finance can become a labour problem months later. A software decision can become

© 2026 Mateusz Chrzanowski · Archestral Studio

an operational dependency. A procurement saving can return as a quality problem. A technical decision can change the behaviour of hundreds of people. A regulation designed for one objective can change the economics of an entire market.

From inside one domain, these effects often appear disconnected. From above, they may be one structure.

That is the perspective I want to develop in this book.

Not a method for turning every reader into a System of Systems Architect. Not a checklist that promises that enough boxes and arrows will produce the same way of seeing. And not a collection of stories from unrelated industries. The material used in this book follows a deliberate boundary. Its analyses are based on public information and public reasoning, while its accounts of realised work are limited to work that can legitimately be presented. When I use a hypothetical construction to demonstrate architectural reasoning, it is exactly that: a hypothetical construction, not confidential client work hidden behind a changed name.

My work with clients is governed by full confidentiality. When a company is optimising its operation or reconstructing its architecture, there is no reason to announce to its competitors that it is improving, what is being changed, how the work is being performed or who is performing it. That information can reveal changes in capability, cost, production, resilience, organisation and strategic direction. Confidentiality is therefore not an administrative detail added after the work. It is one of the foundations of the work itself.

That is why confidential client architectures, internal data, weaknesses, interventions and results are deliberately absent from this book. Their absence should not be mistaken for the boundary of the work. It is the boundary of what should be made public. The function can be demonstrated without turning a client's developing advantage into evidence for somebody else.

The cases matter because the differences between them expose the function. A power plant, a labour market, a company, a technology, a public system or an entire continent can look completely different from the surface. The interesting question is what happens

© 2026 Mateusz Chrzanowski · Archestral Studio

when we stop starting from the surface. Because the visible problem is often only where the system finally becomes visible.

A company reports a loss. A project is late. A market cannot find workers. A cyberattack stops production. A country runs short of infrastructure. Management sees five problems because five departments are reporting them.

Sometimes there are five problems.

Sometimes there is one architecture producing five outputs.

That difference changes everything.

The layer where that difference becomes visible is not a department, an industry or another management level. It is the point at which separate systems stop being analysed separately and begin to form one model.

That is the layer above.

© 2026 Mateusz Chrzanowski · Archestral Studio

Chapter 1

The Problem Started Before You Saw It

The Event Is Late

The moment you begin looking at systems as architecture, one habit has to disappear: the assumption that a problem begins where you can see it. It does not. A factory stops. A customer leaves. A company enters insolvency. A power grid cannot meet demand. A cyberattack paralyses production. A relationship collapses after an argument over dirty dishes. The visible event takes over the conversation because it is concrete. It has a date. It can be photographed, measured, reported and placed on a slide. Somebody can assign responsibility for it. Somebody can announce corrective action. The event is administratively convenient and psychologically satisfying. None of that makes it the cause.

Most serious system failures become visible near the end of their history, not at the beginning. By the time the board, the public, the regulator or even the people operating the system recognise a crisis, the architecture may have been producing that result for years. The visible event is simply the moment when the system can no longer conceal what it has become. If you mistake that moment for the beginning, the response is almost predetermined. A customer left, so find another customer. A factory stopped after a cyberattack, so improve cybersecurity. Electricity demand is too high, so suppress demand. Costs are excessive, so cut costs. A department is underperforming, so replace the manager. A regulation created a problem, so write another regulation. These responses attach themselves neatly to what everyone can already see. They can also leave the architecture that produced the problem almost untouched. That

© 2026 Mateusz Chrzanowski · Archestral Studio

is how organisations spend years solving the same structural failure under different names. They are not repairing the architecture. They are editing its outputs.

Europe's energy debate demonstrates the error with unusual clarity. When pressure on electricity infrastructure becomes visible, attention immediately turns toward whatever happens to be consuming power at that moment. Data centres use too much electricity. Air conditioning uses too much. Heat pumps increase demand. Electric cars need charging. AI requires computing infrastructure. Industry is electrifying. Digital services are expanding. Every visible consumer gets its own debate, and every debate produces another proposed restriction, efficiency standard, reporting obligation, target, label or funding programme. The discussion begins at the wrong end of the system. Europe did not wake up unexpectedly to discover that civilisation had become dependent on electricity. Europe spent decades deliberately moving more of civilisation onto electricity: transport, heating, administration, computing, automation, industrial processes, data infrastructure and increasingly large parts of everyday life. Then electricity demand increased. That is not a surprise. It is the physical consequence of the direction we chose. You cannot electrify civilisation and then pretend to be shocked that civilisation needs electricity.

Yet the latest visible consumer is repeatedly treated as the problem. A data centre appears on the grid and the debate becomes how much electricity data centres should be allowed to consume. That is already the wrong question. The first question is why the energy system was not designed for the demand created by the transformation Europe itself was pursuing. Saying that there is not enough generation capacity does not solve the analytical problem; it merely moves one sentence deeper. Why is there not enough capacity? Why is infrastructure arriving after the demand it was supposed to support? Why did generation, transmission, storage, reserve capability and cross-border coordination not develop as one architecture alongside the electrification they were meant to carry? Once those questions are asked, the data centre stops looking like the cause. It

© 2026 Mateusz Chrzanowski · Archestral Studio

becomes what it always was: one visible output of a much larger system.

Efficiency inside that consumer still matters. Waste remains waste. But making a data centre more efficient does not create the missing energy architecture. The same applies to electric vehicles. If millions of vehicles are expected to charge from the grid, their demand is part of the grid architecture whether policymakers like that fact or not. Charging is not distributed evenly across every minute of every day. People drive, return home, connect vehicles and expect them to be ready again. Demand therefore has structure. If the electricity system cannot support that structure, blaming electric cars is intellectually backwards. Governments, regulators, manufacturers and markets spent years announcing the transition. Infrastructure was supposed to arrive with it. Otherwise the policy eventually becomes absurd: replace your combustion car with an electric one, and then please do not charge it because the electricity system was never prepared for the transition. That is not a failure of the electric car. It is a failure to assemble a transformation into one system.

Europe already has hundreds of cross-border energy infrastructure projects. The number sounds impressive until we remember what a project is. A project is an execution unit. It can build an interconnector, extend a grid, connect offshore generation, create storage or construct another piece of infrastructure. It cannot decide what continental system those pieces are supposed to create. A pile of projects is not a system. It is hope with paperwork. Architecture is the layer that defines why each project exists, what function it performs, what it connects to, what depends on it, what happens when it fails, what compensates for it and what state the whole system is supposed to reach. Without that layer, two hundred projects remain two hundred projects. Administrative proximity does not turn them into architecture.

The contradiction becomes even harder to defend when national decisions are made inside infrastructure that is physically interconnected. A state can choose its own energy mix. It cannot choose

© 2026 Mateusz Chrzanowski · Archestral Studio

to make the physical consequences of that decision stop at its border. Germany can decide what Germany wants to build and what Germany wants to close, but Germany operates inside an interconnected European grid. Closing nuclear capacity therefore cannot be analysed solely as a German policy preference. The architectural question is also what function Germany performs inside the shared system and whether that function remains covered after the capacity disappears. Europe does not need to dictate the exact technology. It needs the architecture to define the function: connected low-carbon capacity, reserve, flexibility, storage, transmission, grid stability or whatever capability the system actually requires. Then a decision to remove firm capacity is no longer made in an architectural vacuum. The country must first know what happens to the function it was supplying.

The same logic applies geographically. Southern Europe, the North Sea, the Baltic, the Alps and other regions do not have identical physical conditions. Their resources differ. Their potential functions inside a continental system therefore differ as well. Treating those characteristics primarily as isolated national portfolios throws away the architectural advantage of an interconnected system. The answer is not another catalogue of projects. The catalogue is not the missing layer. The architecture is. Until that distinction is understood, every new source of demand becomes a convenient target. Data centres use too much. Electric vehicles use too much. Cooling uses too much. Industry uses too much. At some point the argument becomes ridiculous: everything the economy actually needs is accused of consuming the infrastructure that exists to serve it.

If a city builds roads for twenty years without planning for traffic growth, congestion eventually makes the cars visible. That does not mean the cars designed the city badly. Energy makes the same mistake vastly more expensive. Suppressing demand, adding restrictions or asking another consumer to use less does not complete the system. It adds another compensating layer. The next crisis then begins from the architecture left behind by the previous response. Infrastructure remains fragmented. New restrictions accumulate.

© 2026 Mateusz Chrzanowski · Archestral Studio

Businesses adapt around them. Investment moves. Critical capability is built somewhere else. Competitiveness erodes. The system absorbs the consequences until another limit is reached, another visible event appears and another round of surprise begins.

A system can therefore deteriorate while producing an impressive amount of management activity. The two are not opposites. In fact, badly understood systems often generate more and more visible management as their underlying architecture becomes worse. The first serious failure should be the point at which patching stops, because the failure proves something more important than the immediate symptom: the organisation does not understand enough of the system producing it. At that point another patch is not a solution. The architecture has to be reconstructed.

Trigger, Vulnerability, Propagation

The same distinction becomes brutal inside a company. Consider Varta. Its public history offers an almost perfect collection of visible explanations: a cyberattack disrupted operations, a strategically important customer was lost, Chinese competitors became stronger, financial performance deteriorated, restructuring followed and insolvency followed. Arrange those events chronologically and you get a clean story. Clean stories are dangerous because they feel like understanding. This one is not an explanation of the system. It is the chronology through which the system became impossible to ignore.

Take the cyberattack. One digital incident affected production across five manufacturing sites and disrupted operations for weeks. Public information does not establish the internal technical propagation mechanism, and there is no reason to invent one. The operating result is already enough. One incident reached five physically separate production locations. The attacker caused the external event; the architecture determined how much of the productive system could be exposed to its consequence. Whatever the exact internal path was, physically separate plants should not become one uncontrolled failure domain in which a single digital event can disable group productive capability. A cyberattack is not extraordinary.

© 2026 Mateusz Chrzanowski · Archestral Studio

Allowing one to become a multi-site industrial crisis is an architectural result.

This distinction is fundamental: trigger, vulnerability and propagation are not the same thing. The trigger initiates the visible event. Vulnerability determines whether the system can be harmed by it. Propagation determines how far the damage can travel. Compress all three into the sentence “there was a cyberattack” and almost everything worth understanding disappears. The organisation becomes the victim of an external event and the analysis can conveniently stop at the attacker. But the outside world does not owe a company favourable conditions. Competitors do not owe it weak products. Customers do not owe it loyalty. Attackers do not owe it restraint. Energy markets do not owe it low prices. Technology does not owe it stability. Management exists precisely because reality contains pressure. The serious question is not whether pressure will arrive. It is what the system does when it does.

The loss of Apple has to be read with the same discipline. The visible story says that Varta lost an important customer and was damaged by the loss. That chronology is true and still starts too late. The loss of Apple was not the origin of the crisis. It was evidence of an earlier competitiveness failure. Apple did not disappear. The market did not disappear. Demand did not disappear. Another supplier became more attractive. Once that happens, “we lost a customer” is an inadequate diagnosis. The analysis has to move into the architecture behind the offer that failed to retain the customer: product economics, procurement, production, energy, capacity utilisation, labour, quality, development, decision speed and the customer relationship itself. “Apple left” describes the visible event. It does not explain why leaving became rational.

A customer of that scale is not merely a revenue line. It supports production rhythm, capacity utilisation, technical competence, continuity, reference value and negotiating position. Losing such a relationship therefore removes several system advantages at once. By the time the customer moves, the decisive failure has already happened: the organisation behind the offer has failed to remain competitive enough to keep the relationship. Treating the

© 2026 Mateusz Chrzanowski · Archestral Studio

departure itself as the beginning of the problem is simply starting the analysis after the important part is over.

China exposes the same failure on a much larger time scale. Chinese battery and automotive capability did not materialise six months before it became uncomfortable for a European board. It was built over decades. Factories were constructed. Companies expanded. Technology improved. Supply chains deepened. Capital accumulated. Industrial capability accumulated. Manufacturers moved from components toward complete products, from domestic scale toward international markets and from contract manufacturing toward increasingly complete industrial systems. None of this was hidden. What was missed was not the individual event. It was the direction connecting the events.

That failure of integration matters because every fragment can be interpreted correctly and the overall conclusion can still be catastrophically wrong. An acquisition can be treated as an acquisition. A new factory can be treated as a factory. A supplier can be treated as a supplier. A battery company can be treated as another competitor. A Chinese vehicle entering Europe can be treated as an interesting new product. Every description can be factually correct while the system-level meaning disappears. The decisive information is often not contained inside one event. It is contained in the relationships between events.

ATL worked with Apple too. The customer alone therefore cannot explain the difference in trajectory. Capability developed around ATL later contributed to CATL's expansion into automotive batteries. Varta had far more history, established assets, a recognised name, industrial competence and direct exposure to one of the strongest automotive regions in the world. ATL had less. It built more. That comparison is uncomfortable because it destroys the easiest excuse. The story is not that China suddenly arrived. China developed. Europe watched. The information was there. What failed was the ability to read separate developments as one direction of travel.

The same pattern can begin decades before the crisis. A factory is sold and capability leaves with it. A product line is abandoned.

© 2026 Mateusz Chrzanowski · Archestral Studio

A customer relationship is treated as revenue instead of as a platform for developing the next capability. An asset is sold because the transaction makes sense at that moment. A technology is developed but never converted into the product architecture required by the next market. Every decision can be defensible locally. Years later the organisation discovers that the pieces it now needs no longer exist inside the same architecture. Then the crisis becomes visible, and management starts explaining the last twelve months instead of reconstructing the previous twenty years.

Human beings are strongly attracted to temporal proximity. Something happened yesterday, so yesterday must explain today. A customer left, therefore the customer loss caused the crisis. A cyberattack occurred, therefore the cyberattack explains the operational collapse. A competitor entered, therefore the competitor explains the lost market position. Electricity demand increased, therefore the newest consumer created the infrastructure problem. The last event wins because it is closest, clearest and easiest to put into a sentence. Architecture does not work that way. Architecture has memory. Today's output contains yesterday's decisions, the decisions before them and sometimes decisions made by people who left the organisation years ago for reasons nobody even remembers. The people leave. The dependency does not. The layout does not. The contract does not. The software does not. The missing capability does not. The consequence waits.

That delay creates one of management's most dangerous illusions: a structurally bad system can produce good numbers for a long time. A company can be profitable, sales can grow, production can increase and customers can continue buying while the architecture deteriorates underneath. Profit proves that the system can currently produce profit. Nothing more. It does not prove that the architecture is resilient. It does not prove that dependencies are understood. It does not prove that employees are not compensating for broken processes. It does not prove that costs are not being transferred between functions. It does not prove that the customer base is secure, that the

© 2026 Mateusz Chrzanowski · Archestral Studio

digital backbone can survive a serious disturbance or that a competitor is not building the next market while the incumbent is optimising the current one. Treating good current output as proof of healthy architecture is one of the most expensive mistakes a board can make.

Bad architecture can survive for years when something else is paying its bill. Margin can absorb inefficiency. Growth can hide waste. Employees can compensate for defective procedures. A strong customer can hide weakness elsewhere. Cheap capital can conceal poor economics. Excess capacity can cover bad planning. Market leadership can disguise slow decisions. The company appears to work because buffers are quietly paying for what the architecture failed to solve. When the buffer disappears, the problem seems to arrive overnight. It did not. The bill arrived overnight. The architecture accumulated it slowly.

The natural response to the visible crisis is then to repair what is visible, and that response can make the architecture worse. Another procedure is added. Another reporting layer appears. Another approval is required. Another emergency supplier is contracted. Another software platform is installed. Another restriction compensates for missing capacity. Another restructuring programme removes cost without reconstructing why the cost exists. The symptom becomes smaller, management announces progress and the underlying system gains another compensating layer. A symptom-only response is not neutral. It becomes part of the architecture.

That is why the first crisis is so often followed by another. The first crisis is the first moment when the architecture becomes loud enough that nobody can ignore it. If the response stops at the symptom, the organisation returns carrying both the original defect and the structure added to compensate for it. Repeat that process often enough and the organisation eventually becomes a sedimentary structure of old problems, old fixes and fixes for the fixes. Nobody can say with confidence which layer performs a necessary function and which exists because something else failed years ago. The next crisis is not mysterious. It is accumulated architecture becoming visible again.

© 2026 Mateusz Chrzanowski · Archestral Studio

When Nobody Sees the Whole

Human systems make the same mechanism harder to deny because the number of interacting variables becomes enormous. Imagine two people living together. One comes home, sees dirty dishes in the kitchen and an argument explodes. If we begin and end with what we can see, the explanation is trivial: the dishes caused the argument. The explanation is also absurd. That person may already be ill, exhausted or under intense stress. Work may have gone badly. A boss may have shouted at them. Traffic may have been terrible. A queue may have consumed the last of their patience. A bag may have torn in a car park and the groceries may have landed on the pavement. By the time the person enters the house, the nervous system is not in the state in which it left that morning. Then there are dirty dishes. The dishes are real. They are not an adequate explanation. The same plates on another day might produce nothing because the physical input is the same while the state of the system is completely different.

Now add the second person. Perhaps home normally functions as the place where accumulated tension disappears. The partner listens, provides safety and absorbs part of the pressure. But that evening the partner is also exhausted. There is trouble at work, uncertainty about the future, financial pressure, ambition, lack of rest or some other accumulated load. The person who normally absorbs tension has no capacity left to absorb it. One irritated sentence meets another. The dishes become the ignition point. They still are not the cause.

Go further back and the system expands again: childhood, family patterns, previous relationships, learned responses to conflict, fear, financial security, health, legal conditions, economic conditions and the environment in which both people are living. The visible argument can sit on top of thousands of interacting relationships. A human system is not conveniently binary. X does not always produce Y. Sometimes X produces Y only in a particular state. Sometimes X and Y together produce Z. Sometimes Z changes whether X matters at all. Reducing that system to the final trigger is not simplification. It is analytical destruction. The dishes are visible. The

© 2026 Mateusz Chrzanowski · Archestral Studio

architecture of the day is not. The argument is visible. The accumulated state of two human beings is not.

Companies perform the same intellectual trick with people. An employee makes a mistake, so the employee becomes the explanation. A manager fails to deliver, so the manager becomes the explanation. A team misses a target, so the team becomes the explanation. Sometimes the person genuinely lacks the required competence. People are not equally capable, and pretending otherwise is useless. But finding a person at the end of a failure chain still tells us nothing about what information reached them, what authority they had, what workload they were carrying, what tools they were given, what incentives acted on them, what defects they had already been compensating for or why the architecture placed them in that position. Blame is easier because a person has a name. Architecture usually does not. Removing the person therefore produces immediate psychological relief: someone has been held responsible, management has acted, the box has been ticked. Then the replacement enters the same system and everybody waits to see whether the architecture can manufacture the same failure again.

Part of the problem is cognitive. Most people do not naturally hold large numbers of interacting layers inside one model. That should be an ordinary observation, yet organisations routinely behave as if admitting differences in cognitive capability were somehow an attack on human value. It is not. Human beings are equal in dignity. They are not equal in cognitive capability, memory, spatial reasoning, mathematical ability, pattern recognition, language, manual ability, social intelligence or capacity to hold many interacting variables simultaneously. Those are factual differences in capability, not a hierarchy of human worth.

Confusing those two ideas creates a serious organisational weakness. If admitting that another person understands a particular class of problem better than I do feels like admitting that I am a lesser human being, I gain an incentive to pretend. I defend conclusions after my competence has run out. I treat asking for help as loss of status. I interpret someone else's stronger capability as a threat. Give

© 2026 Mateusz Chrzanowski · Archestral Studio

that behaviour a management title and the problem acquires authority. A title can become a mechanism protecting ignorance.

Systems do not care about titles. A dependency does not disappear because the CEO cannot see it. A technical relationship does not become simpler because the board prefers a five-slide summary. A production problem does not reshape itself around the competence boundary of the person assigned to analyse it. Reality does not negotiate with ego.

We already accept this in domains where the consequences of pretending are too serious to tolerate. Aviation, aerospace, defence and critical infrastructure do not assume that management authority somehow creates mastery of every subsystem, technical discipline and dependency inside a sufficiently complex architecture. Specialists provide depth inside individual systems. Somebody still has to hold the architecture connecting them.

In these fields, this responsibility is formally recognised. System of Systems Architect is an existing professional position. That matters because business has no exemption from complexity. A company can contain production, finance, logistics, software, infrastructure, people, suppliers, energy, information, physical space and dozens of other interacting systems, yet still behave as if authority over the organisation automatically creates visibility of the whole. It does not. A board can spend years governing a system it has never reconstructed as a whole. A consulting firm can interview management, analyse reports, review procedures and produce recommendations without ever reconstructing the operating architecture beneath them. A regulator can respond to one visible pressure with another rule without understanding the system producing it.

Everyone can be intelligent, experienced, professional and senior. The architecture can still be wrong. At some point somebody has to say: we do not see enough. That is not an admission of failure. It is the first intellectually useful conclusion available once the limits of the current explanation become visible. It changes the question from What should we do about the problem? to Who can reconstruct what we are not seeing? The first assumes the system is understood and only action is missing. The second admits that

© 2026 Mateusz Chrzanowski · Archestral Studio

the explanation itself may be defective. Those are entirely different starting points.

The same rule applies to me. System of Systems Architecture does not mean omniscience. When I enter a production environment, I do not assume that operating at the Layer Above makes me more knowledgeable about every machine, material and operation than the people who work with them every day. That would be idiotic. I ask. I want the engineer's knowledge of the mechanism, the operator's knowledge of the workstation, the maintenance technician's knowledge of failure, the warehouse worker's knowledge of physical flow and the quality inspector's knowledge of recurring defects. Formal rank is irrelevant to the value of the information. Reality is not organised according to the organisational chart.

The operator may know exactly where a formal procedure stops matching physical work, which material variation creates trouble or what has to be corrected manually when the machine behaves differently from its documentation. Ten people operating ten nominally identical machines may be living inside ten materially different operating realities. One machine fails differently. One operator has developed a workaround. One receives different material. Another knows that one official step is routinely ignored because following it literally would make production impossible. None of that knowledge becomes less important because it belongs to someone lower in the hierarchy.

My function begins when those fragments have to become one structure. I can take the engineer's explanation, the operator's experience, the warehouse worker's observation, the manager's decision, the financial consequence, the information path and the technical dependency and hold them inside the same architecture. That does not diminish the specialists. It makes their knowledge usable beyond the boundaries of the fragment in which it was created. The engineer may know more than I do about a particular mechanism. The operator may know more about a workstation. Finance may know more about a specific accounting mechanism. Legal may know more about a particular legal constraint. None of that gives any one of those functions automatic visibility of the whole.

© 2026 Mateusz Chrzanowski · Archestral Studio

The error is believing that because engineering understands the machine, engineering understands the company; because finance understands the financial result, finance understands what produced it; because management receives every report, management sees the organisation; or because the regulator understands the regulation, the regulation explains the system. Every participant can be competent inside their own field and the whole organisation can still be wrong.

That is what makes complex failure difficult. There is often no single idiot to blame. There may be hundreds of competent people making locally rational decisions inside fragments they genuinely understand. The fragments simply do not assemble themselves. A technically correct decision can damage the financial layer. A financially attractive decision can destroy operational resilience. A locally efficient process can increase total system cost. A national energy decision can weaken a continental structure. A customer strategy can protect today's revenue while destroying tomorrow's capability. The system-level result can be disastrous even when many individual decisions looked defensible when they were made.

That does not remove responsibility. It moves responsibility to the level where the real failure exists. If people are acting reasonably inside a badly constructed architecture, replacing them will not repair the architecture. Another management team can inherit the same system and manufacture the same class of output. Another transformation programme can reorganise the same defect. Another regulation can be placed on top of the previous regulation. Another catalogue of European energy projects can be announced while the continent still lacks the architecture that gives those projects one coherent function. Activity is not repair. Replacement is not reconstruction. Seniority is not system visibility.

A Better Story Is Still Not a Model

Visible problems are dangerous because they compress enormous complexity into sentences that everyone can understand. The factory stopped. The customer left. The numbers fell. The employee failed. The grid is overloaded. The data centre consumes too much

© 2026 Mateusz Chrzanowski · Archestral Studio

electricity. The couple argued about dishes. Every sentence can be factually correct. Every sentence can also be almost useless as an explanation.

That does not mean the visible event should be ignored. If a factory stops, restore production. If a customer is leaving, protect the relationship. If the power system is under stress, preserve continuity. If two people are screaming at each other, deal with the immediate conflict. Architecture is not an excuse to stand beside a burning building and lecture everyone about the history of fire. Operational containment is necessary. But containment and understanding are different tasks. The emergency tells you what must be handled now. It does not tell you what created the emergency.

The problem is what happens after the immediate pressure falls. Organisations love to stop there. Production restarted: problem solved. Customer replaced: problem solved. Financing secured: problem solved. Demand reduced: problem solved. New procedure introduced: problem solved. No. The event was handled. That is not the same thing as repairing the architecture. The serious questions begin after the emergency: what conditions existed before the event, which decisions created them, which dependencies connected them, which buffers hid them, which earlier events were already signals, why the trigger travelled as far as it did and why the system failed to contain it.

Those questions move the analysis backwards from the event, then sideways, downward and across the boundaries of the category in which the event first appeared. Very quickly, the original problem stops looking isolated. The cyberattack stops being only cybersecurity. The lost customer stops being only sales. The energy shortage stops being only generation capacity. The employee mistake stops being only an HR issue. The argument stops being about dishes. The visible category was merely where the output appeared. It was never proof of where the problem lived.

Language makes this worse because organisations name problems before they understand them. “Cybersecurity incident.” “Commercial issue.” “Labour problem.” “Energy shortage.” “Supply-chain disruption.” “Management failure.” “Employee performance issue.”

© 2026 Mateusz Chrzanowski · Archestral Studio

The label creates a box. The box gets an owner. The owner gets a scope. The scope determines who enters the room. Before anybody has reconstructed the system, the organisation has already decided what kind of problem it is. That is not neutral vocabulary. It is a mechanism for trapping the analysis inside the visible layer.

The first description therefore deserves suspicion. It may be true and still be radically incomplete. The place where the system screamed is not necessarily the place where it was wounded. A factory can scream through IT after years of dependency accumulation. A company can scream through insolvency after years of strategic and operational deterioration. A continent can scream through electricity scarcity after decades of disconnected infrastructure decisions. A human being can scream over dirty dishes after an entire day, year or lifetime of accumulated state. The final event gives the system a voice. It does not give you its architecture.

Recognising that creates another trap. Once the obvious explanation looks too shallow, people search for something that sounds deeper. Not the cyberattack – dependency. Not the lost customer – cost. Not data centres – generation capacity. Not the dishes – stress. The language becomes more sophisticated, but the intellectual mistake can remain exactly the same. Move one layer below the symptom, find something plausible, give it a more intelligent name and stop. Complex systems can support many plausible stories. A plausible story is still only a story until the dependencies that produce it have been reconstructed.

Once we accept that the visible event is usually near the end of the causal history, we lose the right to guess what came before it. We have to know. That requires something harder than a better explanation.

It requires a model.

© 2026 Mateusz Chrzanowski · Archestral Studio

THE SAMPLE ENDS HERE.

Continue reading

The Layer Above

The complete book continues with Chapter 2.
Available on Amazon UK.

GET THE COMPLETE BOOK

Amazon UK