LABRYS FRONTIER SERIES

Systemic Approach to Technology (SAT):

The case for society-focused technology

October
2023

https://doi.org/10.59262/tm58mc

Systemic Approach to Technology (SAT):

The case for society-focused technology

Labrys

Mariana Cunha e Melo
Jonas de Abreu

Labrys's mission is to connect technology, business, and policy to build, inspire, and enable society-focused technology at scale. Our ultimate goal is to create a world where technologists can develop society-focused technology, citizens can effectively debate what society needs from technology, and regulators can align incentives to strengthen an open, inclusive, and sustainable economy for all.

Center for Technology and Public Interest, SL
Carrer de Bailèn, 11, Barcelona, Spain, 08010
www.wearelabrys.com

Table of contents

Summary

This article introduces the Systemic Approach to Technology (SAT) as a framework for society-focused technology. The authors argue that technology should be teleologic, meaning it should be evaluated based on its application rather than being seen as inherently good or bad. They emphasize the importance of understanding the systemic relationships between technology and its context, advocating for a systemic approach to technology design. The paper also discusses the need for technology development to be propositive and focused on addressing societies' greatest challenges. The authors highlight the potential of this approach in guiding a new era of technological development and shaping a positive impact on society.

I. Introduction

When we initially established the Center for Technology and Public Interest, our objective was clear: to leverage our knowledge and diverse backgrounds to find comprehensive solutions to problems that couldn't be adequately addressed through a narrow, specialized approach. Over time, we came to realize that the solutions we had developed throughout our careers reflected a fundamentally different approach to technology compared to what we observed elsewhere.

While some aspects of our approach align with what Bruce Schneier refers to as public interest technologists, there are numerous other elements that extend beyond that conceptual label. Now, we take this opportunity to elaborate on the guiding principles we believe should shape

a new era of technological development, as well as to outline our aspirations for the future work of CTPI.

We decided to put our approach into words because we believe it to be crucial that we, as a society, shift our perspective on technology - how we perceive it, interact with it, and build it. Many people view specific technologies as the sole saviors or destroyers of humanity. As if technology was itself good or bad, capable of kindness or evil. Industry leaders prioritize what is trendy over what is urgent. Technologists have become fixated on bringing decades-old science fiction to life instead of addressing the pressing needs of humanity today. An astonishing amount of talent, time, and resources are wasted on narrow solutions that fail to consider the systemic relationships between problem elements and the fundamental components of effective solutions.

Meanwhile, we're at a momentous chapter in the history of our generation. We are, sociologically speaking, at the turn of a cycle of our civilization, which is obviously challenging but is also filled with opportunities for transformation.

In Western culture, we have grown accustomed to viewing history as a linear process, from genesis to the final judgment. So, we're hardwired to associate the current climate of multiple simultaneous crises with the apocalypse. However, both nature and society experience time in cyclical patterns, from the passing of days and years to changes in the seasons and the cultural cycles of our civilization. From The Turning Point, p. 26:

> "Western sociologists have confirmed *[the Chinese]* ancient intuition *[that crisis is an aspect of transformation]*. Studies of periods of cultural transformation in various societies have shown that these transformations are typically preceded by a variety of social indicators, many of them identical to the symptoms of our current crisis. They include a sense of alienation and an increase in mental illness, violent crime, and social disruption, as well as an increased interest in religious cultism–all of which have been observed in our society during the past decade. In times of historic cultural change these indicators have tended to appear one to three decades before the central transformation, rising in frequency and intensity as the transformation is approaching, and falling again after it has occurred. (...)
>
> Cultural transformations of this kind are essential steps in the development of civilizations. The forces underlying this development are complex, and historians are far from having a comprehensive theory of cultural dynamics, but it seems that all civilizations go through similar cyclical processes of genesis, growth, breakdown, and disintegration".

The world is currently in crisis, and significant transformations are on the horizon. But this doesn't mean it's the end of the line. It means the turn of a new cycle. Acknowledging that we're entering a new cycle means recognizing the unique opportunity to recognize our responsibility to use this transformational energy to the benefit of society.

It's time for a new approach to technology – one that is inherently *systemic* and focused on its application and intentional purpose.

This document represents our initial attempt to define the characteristics of our systemic approach to technology.

II. The three pillars of the Systemic Approach to Technology

II.1. Our approach to technology must be teleologic

In recent years, public opinion has been bombarded by the misguided notion that technology is an end in itself and that specific technologies will lead humanity to salvation or damnation. That concept has lived in the popular imagination for ages, but in the last few decades, humanity has lived short cycles of greater and milder intensity. Perhaps with greater intensity since the creation of the atomic bomb, when humanity faced for the first time the real existential threat at the hands of science and technology. Lately, we've seen people putting their hopes or fears into technologies like Artificial Intelligence, Metaverse, Blockchain, etc.

Even within the tech industry, in the past few years, executives time and again expressed the view that their technology is inherently good, regardless of its specific use or application. For example, executives of social media platforms often argue that social media is inherently good because it connects people and promotes free speech. Blockchain enthusiasts believe that blockchain is the solution to almost any problem, from supply chain management to voting systems. Similarly, people immediately bought into the idea that the metaverse would revolutionize the way we interact with one another and with technology.

That's what's been called *technological solutionism*, a term coined in 2013 by the American writer Evgeny Morozov. To be fair, we aren't nearly as skeptical as Mr. Morozov. But the fact of the matter is that technology by itself does not have the power to do good or bad. It's the use humans employ and its consequences that can have any kind of moral charge. This techno-solutionism approach is counterproductive even for those who employ it. It invites debate into the ethics of technology itself and the discussion about whether technology is dangerous or harmful in itself. It deepens the public debate about whether specific technology should be banned or further regulated.

That sort of binary thinking is cultivated in the tech industry, spreads to public opinion, and bubbles up in discussions around public policy. One recent example of that phenomenon can be found in the concerns about mental health issues related to kids' abuse of social media,

which gave room to proposals that severely limit freedom of expression and information online. This case is especially interesting because it showcases traditional companies finding ways to push to technology the responsibility for the toxic environment they inhabit. It is as if the problem lies solely on how publicly kids propagate impossible beauty standards while the source of these very standards and the causes behind women's body shaming safely slide into the background.

To position specific technologies as saints or demons is to miss the point. Technology is not in itself good or bad. Neither hero nor enemy. It can only be defined and evaluated in the context of its employment by humans. It's the humans behind the technology and the activity they exercise that should be under scrutiny, not the technology itself. George Orwell's book "1984" is a cautionary tale about state authoritarianism, not camera technology.

The issue of the ethics of a technique is a misplaced debate. We cannot truly assess the potential impacts of a new technology, nor can we steer the course of technological development toward meaningful ends if we are either in love or terrified of a technique. Technology is not an end in itself. The application of a piece of technology is important not only to assess where it stands from an ethical point of view but also from a technical point of view. Out of the context of a specific application to a certain end, technology is pointless. The application context is an intrinsic part of what a piece of technology *is*, whether it *works*, and whether it *is good* in an ethical sense. That's what philosophers call *teleologic*: the property of

something that can only be understood and *valued* by analyzing its purpose.

Technology is neither good nor bad, but the uses humans employ make it so. It can be either a prison or the door to infinite space, depending on our dreams, as The Bard would say (and here we mean Shakespeare, not Google). It's time to turn our focus back to the human actions we want to embrace or restrict in this upcoming new cycle of humanity. It's time we put in proper debate what our dreams for *technology applications* are, what they *should be*, and *how* to achieve them.

II.2. Our approach to technology design must be systemic

a. Systems reasoning

In philosophy, systemic reasoning opposes the mechanistic thinking that is the front and center of the traditional scientific method, as articulated by Descartes and furthered by Newton four hundred years ago. Fritjof Capra describes this process majestically in the 1982 book The Turning Point. According to Capra, at the core of that paradigm is the idea that the cosmos and everything in it works like a mechanical clock. So, to understand reality and advance human knowledge, one should break down the complex whole into smaller parts. The underlying assumption from this approach is that the whole is the exact sum of its parts, and the parts are material indivisible unities (literally, *atoms*). This cartesian thinking was the basis of the

scientific method that boosted the scientific revolution in Century XVII.

However, as Capra continues, both science and scientific thinking have changed dramatically in the last century, from the evolution of thermodynamics to the foundation of the theory of relativity, the principle of uncertainty, the incompleteness theorems, and modern quantum mechanics. The mechanic clockwork is no longer able to represent our perception of the cosmos. Neither does Descartes' cartesian thinking. A new paradigm has emerged from the confrontation with the inherent complexity of reality and the acceptance that simplistic models and unsound deterministic phantasies could not continue to help us advance human knowledge (for more details, see The Turning Point, Chapter II).

The new paradigm relies on systemic reasoning, complexity, and recursive thinking.

Thinkers historically dispute inductive and deductive reasoning. According to Popper, the difference between inductive thinking and deductive thinking is that, with inductive thinking, we generalize the specific instance we can observe as a general rule while with deductive thinking, we conceive the specific instances we can observe as manifestations of an underlying system.

Taking this idea to the next level, in what we can call *recursive thinking*, we create models to describe the underlying system, its relationships, and its potential elements in a way that the specific observation can be produced. The correctness of the model is then constantly

tested against new observations, and once an observation is made that is incompatible with the model, the model is adapted or replaced by one that could generate the new observation as well.

The exciting part is that, from this approach, it makes no sense to say that the model was right and then it was found to be wrong. Systems are not static entities, so our models should be in a constant evolution process (On Complexity, p. 50). The feedback loops between elements of the systems and their relationships to each other and the whole put every system in constant change (entropy). So, recursive thinking must also constantly challenge and modify our models to the events (phenomena) we observe (On Complexity, p. 14).

In the systems theory, neither the systems' elements nor its whole is the actual object of analysis. The fundamental unit of a system is the relationships between its elements. That means one does not need the completeness of the whole to understand its parts. Nor is it possible to understand the whole by looking at its parts. It's only by understanding the relationships between the elements of a system that one can truly understand a system and predict its behavior patterns. We do this by working backward from the event we can observe to their underlying premises. We call this process investigating the *premises* of any given event.

A premise is a set of logical assumptions that explain why something is the way it is. Each event may have multiple premises, and, in each system, there are a multitude of events with their own premises. It's the compatibility of all underlying premises that make up the model of a system.

The system itself, on the other hand, is the set of elements and interrelationships that "produce their own pattern of behavior over time" (Thinking in Systems, p. 2).

But how do we apply this system reasoning to the tech industry?

Clayton Christensen hinted at this kind of thinking in Seeing What's Next. The book makes the case for a theory-based approach to decision-making. It instructs analysts to understand the inner workings of a market, its elements, and forces to predict behavior and make sound strategic decisions. From Seeing What's Next, p. xx-xxi:

> "Consider analysts working for a large investment bank. How do they predict industry change? Typically, they gather historical data, determine trends, and make projections. They extrapolate a firm's past earnings to determine its future cash flow and then discount that cash flow at a risk-adjusted rate to determine firm value. (...) In doing so, they rely on an implicit theory: The past is a good predictor of the future.
>
> Now consider management consultants, seeking to tell a company how to organize its sales force. Many of them approach this challenge by identifying a "best-practice" company and gathering gigabytes of data "proving" how the company's particular sales force design is key to its success. If their client would only imitate that comparison company, the consultants say, they too would reap the rewards. Again, the consultants base their recommendation on an implicit assumption: Companies find success when they mimic

> actions taken by successful or "excellent" companies.
>
> Sometimes these assumptions are correct and lead to great insight. But sometimes they don't. **The past is a good predictor of the future only when conditions in the future resemble conditions in the past.** And what works for a firm in one context might not work for another firm in a different context".

The *conditions* and *context* Christensen mentions are the parts and interrelationships between those and the system. Christensen's Disruptive Innovation Theory, Resources, Processes, and Values Theory, Value Chaim Evolution Theory, and Conservation of Integration Theory are powerful because they work on an incredibly high level of abstraction, explaining how the very basic elements of a market behave when exposed to specific forces. It's what Donella Meadows calls feedback loops, which influence the flows of stock in a system. Meadows uses far less abstract language to talk about systems, which is ironic, in a way, since systems theory allows us to operate in higher degrees of abstraction. But also makes a first contact with systems theory far more intuitive for newcomers. From Thinking in Systems, p. 17, 18, 24, 25:

> "A stock is the foundation of any system. Stocks are the elements of the system that you can see, feel, count, or measure at any given time. (...)
>
> Stocks change over time through the actions of a flow. Flows are filling and draining, births and deaths, purchases and sales, growth and decay, deposits and withdrawals, successes

> and failures. A stock, then, is the present memory of the history of changing flows within the system. (...)
>
> There is one more important principle about the role of stocks in systems, a principle that will lead us directly to the concept of feedback. The presence of stocks allows inflows and outflows to be independent of each other and temporarily out of balance with each other. (...) When a stock grows by leaps and bounds or declines swiftly or is held within a certain range no matter what else is going on around it, it is likely that there is a control mechanism at work. In other words, if you see a behavior that persists over time, there is likely a mechanism creating that consistent behavior. That mechanism operates through a feedback loop. It is the consistent behavior pattern over a long period of time that is the first hint of the existence of a feedback loop. A feedback loop is formed when changes in a stock affect the flows into or out of that same stock".

Finally, this kind of thinking is not too dissimilar to what is called second-order thinking. Typically, second-order thinkers are looking for different levels of consequences: direct effects of decisions (first-order) and the direct effects of those first set of consequences (second-order). What we describe as the investigation into the premises of any given event in a system is similar to what we could call finding the second-order *causes* of any given phenomenon. That is, it's the most fundamental logic of what makes a system evolve or, in Meadows' words, *flow*.

Those are hints of a systemic approach to business. Developing that approach further is the key to our new approach to guide how we think, interact, and build technology. The Systemic Approach to Technology comes into play throughout the whole product lifecycle. From conception to the analysis of results and optimization, systemic thinking can help engineers, product managers, designers, and executives make more consistent and reliable decisions. When applied by regulators and other policymakers, systemic thinking can help them understand the underlying forces that act in the market and what are the levers they can pull to influence behavior toward the common good. Finally, once citizens and society at large learn to connect the dots and understand the base code of how technology can be used to affect change, we'll be better able to discuss what we want from technology and influence the pace and path of development more intentionally.

To do this, we must practice seeing all events we witness as manifestations of an underlying system. That is, to see each event as the consequence of a series of interactions between elements in a given system. If, for example, the competition level in the financial services industry in country W is low, the systemic approach would require us to try and understand which elements of that market can influence the level of competition and how they interact with each other.

The fundamental question is: what are the premises of this event? Every time we ask that question, we must contrast our answers with those we asked before to ensure they are

all compatible. This investigation into the premises of an event can stretch wide when we investigate multiple causes for the same event or deep when we look into the logical assumptions for a given premise (the premises' premises). In doing so, we build a mental model of a system that could, given the circumstances, lead to the event we are analyzing (a low competition level in country W). From that moment on, every new event that presents itself should be compared against this mental model to assess whether the version of the system we built in our head could also output the new event. If yes, great. If not, we must adjust the mental model of the system so that it can output all the observed events.

Once the model for the system and its inner workings is created, it's possible to make an alternative scenario analysis by changing one or more elements of the system and trying to predict the changes it might have in the other elements and in the way they interact with each other. We can apply this kind of thinking to make reactive predictions about the potential impact of new legislation that comes into force or a new product that is brought to market, looking for what Clayton Christensen calls "signals of change." The same goes for the analysis of the potential impact of new technology applications. That being said, there are two other ways of employing systemic reasoning in the context of business, public policy, and technology.

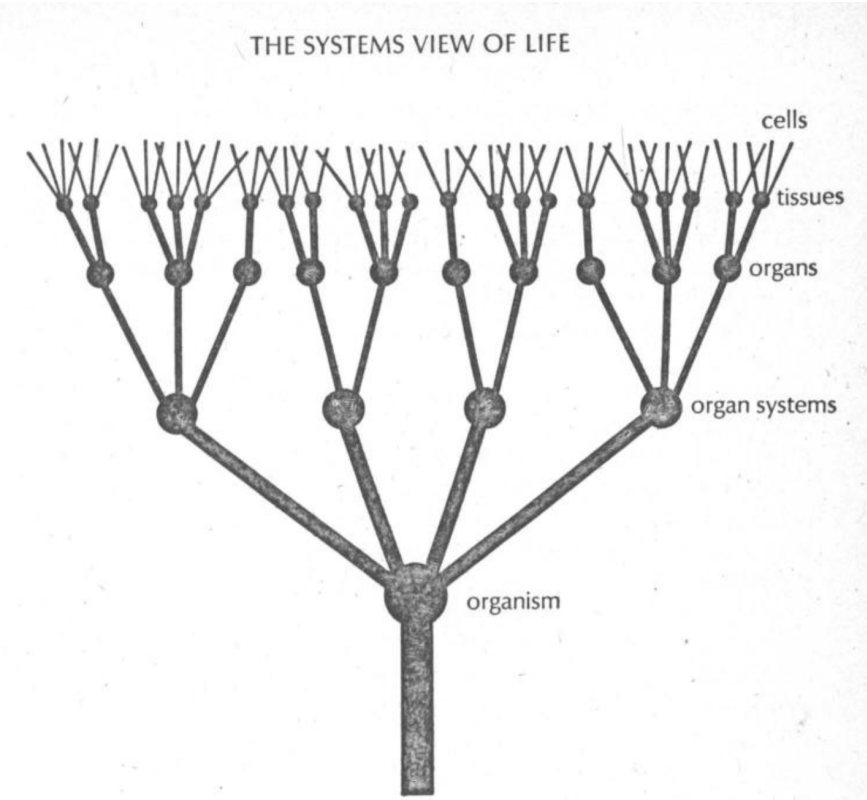
The second way is to proactively identify opportunities to make interventions in a given system. Take the example of the competition level in country W's financial service industry. Given the circumstances of country W, it could be

the case that the regulatory burden to start operating any kind of financial service is too high and that creating a path for newcomers to start small and grow in complexity as they grow in relevance in the market could ignite a wave of transformations in the country W's financial system. Product teams and executives can also use systems reasoning to make strategic decisions after modeling the system of the market in which they operate (for an insightful quick start, we suggest reading Seeing What's Next in full).

Finally, the third way of using systems reasoning in this context is to design a new system from scratch as a means to make an impact in a broader system. We'll go through this concept in greater detail in the section about systems-driven design. But here is where technology design can be leveraged to pull the broader systems' levers and affect change. The ultimate example of such an impact is Tim Berners-Lee's inventions of the World Wide Web, HTML language, URL system, and HTTP protocol. In short, TimBL created the primitives and elements that, combined, enabled every single internet-based innovation ever since.

Here, it's worth making a quick conceptual note about how systems theory and computer science can relate. When we discuss different levels of abstraction in a system, we are operating on the plane of logical relationships. We identify an event (phenomenon) and work backward through its premises. As we do this, we work from a greater level of complexity to a lower level. And, by doing so, we identify elements and forces that are simpler in nature but more powerful in terms of what they can create by interacting

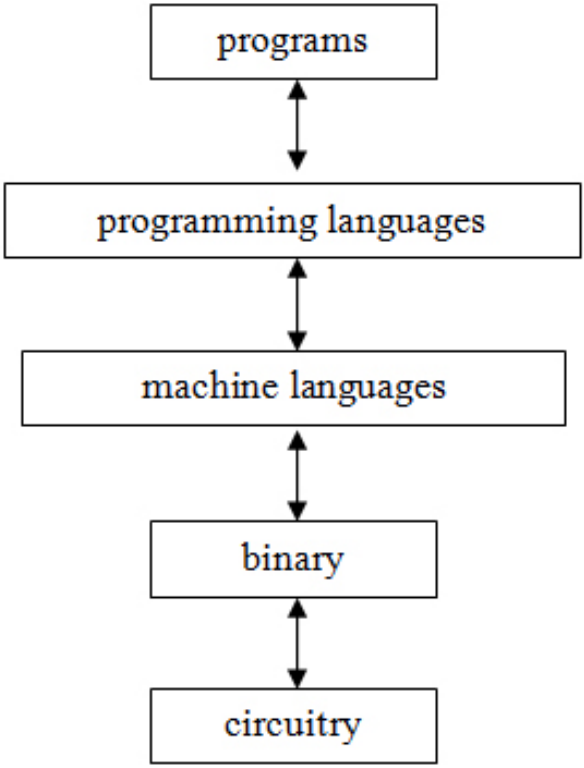
with other elements or forces. A good visual example of this is found in nature, but it's also true for social, psychological, economic, or any other kind of system. See the image below from The Turning Point, p. 281:



Systems tree representing various levels of complexity within an individual living organism.

As we move toward a greater degree of abstraction and lower level of complexity, the potential for combination and creation of new things also increases. Now, let's go back to computer science. In one sentence, what we find is that abstraction in computer science is the process through

which a simpler, more general language is used in a way to create a more specific language and allow programmers to ignore how the more general language works (see Abstraction in Computer Science). The image below illustrates well how this works. From The Semiotic Abstraction:



Once the binary language was created, programmers were able to ignore how the physics of circuits work so that they could focus on an easier language and create new things. Yet, binary is still fairly hard to operate, so computer scientists continued working upward the level of specificity of languages until they were able to use language that

humans can understand (programming languages), reducing the difficulty of programming. Still, not all people learn programming languages. So programmers write even more specific applications of these languages to create the programs and interfaces that billions of people can use. When a person uses their phone to read an article, they are provoking actions in the circuit of their phones even without knowing a first thing about how computers work.

We can say the circuitry level of computer science is more powerful than any program in existence because, in the end, *all programs are the result of a sequence of circuit operations*. So, we can do *more things* with circuitry than with any given program. At the same time, although binary language is *harder* for us to understand, it is simpler in *nature* than a given program because a program incorporates considerably more details, nuances, and, therefore, *complexity* than binary language does. The best way to visualize this logic is through the concept that binary language has literally just two elements (1 and 0), and the Turing Machine describes only six types of fundamental operations in machine language (primitives) and is able to describe all computing operations broadly implemented today. It's *harder* to operate, but it's *simpler*, *more abstract*, *flexible*, and *powerful* than any programming language or program.

A fun fact may help illustrate the point about the greater complexity of programming languages compared to machine languages or binary. Almost everything programmers write in a programming language will eventually be thrown away in the process of translating into

machine language (a process called *compiling*, which literally means to *reduce*). This happens because programmers need way more semantics to understand/change what some code is doing than the computer does to execute operations. But context doesn't matter for computers; it only matters to programmers. It's unnecessary *complexity* at their level of abstraction.

Now, applying the language from systems theory, we have that programs are events we can analyze backward from greater complexity to more powerful elements to find their premises in different degrees of abstraction and complexity until the physical operations of the hardware. The same logic is applicable to all kinds of computing problems. See, for instance, the OSI Model, which describes the different levels of abstraction in how the internet works. From OSI Model:

OSI model

Layer			Protocol data unit (PDU)	Function[27]
Host layers	7	Application	Data	High-level protocols such as for resource sharing or remote file access, e.g. HTTP.
	6	Presentation		Translation of data between a networking service and an application; including character encoding, data compression and encryption/decryption
	5	Session		Managing communication sessions, i.e., continuous exchange of information in the form of multiple back-and-forth transmissions between two nodes
	4	Transport	Segment, Datagram	Reliable transmission of data segments between points on a network, including segmentation, acknowledgement and multiplexing
Media layers	3	Network	Packet	Structuring and managing a multi-node network, including addressing, routing and traffic control
	2	Data link	Frame	Transmission of data frames between two nodes connected by a physical layer
	1	Physical	Bit, Symbol	Transmission and reception of raw bit streams over a physical medium

Technologists routinely choose to operate in different levels of abstraction depending on how much flexibility they need to solve a problem: the more flexibility and combinability they need, the more abstract and less complex the layer they go to. The same operation can be performed outside the strictly technical work of a technologist. When building

a solution to any given problem, it's the technologist's job to understand the premises and most primitive aspects of the demand at hand. And then to find the *simplest* way to model the solution to the problem in a way that can preserve to the maximum extent possible the flexibility of a greater abstraction level so that it can solve new demands from the same system in the future, as they evolve.

For that to happen, we need to take specific demands for use cases as manifestations of an underlying system and take a higher degree of abstraction to identify its most basic elements and learn how they interact with each other. And then, create a model that could represent the elements and their interactions to figure out what levers could be used to bring equilibrium to the system. Finally, it is the technologists' job to model the underlying system in a way that primitives and rules can be used to describe not just the solution to the demand at hand, but an unknown number of other demands that can result from the same system.

b. Technology is contextual, depends on domain expertise to be useful, and should always be problem-driven.

Our industry is growing increasingly specialized in pieces of technology. Software engineers fall in love with programming languages, architecture models, methods, and tools and become highly specialized in those techniques while falling further away from the domain they are operating in. And for a good reason. Companies expect their workforce to be specialized, and the technical

community is very good at creating silos of practitioners of this or that technical trend. Courses sell trends as the big solution to all problems, and millions of technologists follow along.

So that when engineers go out in the world and start building things, they look for applications of the tech they know. They look for the things that they know and that could be applied to the reality around them. By doing so, they operate in an extremely reduced level of abstraction and try to fit the demands they face into the solutions they've already got. Companies reinforce that behavior by hiring people who are highly specialized, to the point of comic absurdity. That is a technology-first approach that considers what can be done and then looks for what can be sold from the pool of solutions they already have.

Companies are concerned about what can be done today to create value for customers and return to stakeholders, as they should. But that doesn't mean a technology-first approach is a way forward for all companies in the long run. And it definitely does not mean that should be the way we, the society, should deal with technology.

Even in a customer-first approach, those who advocate for a more well-thought-out product discovery process are focused on deciding *what* to build. That focus has the merit of preventing product teams from trying to solve the hardest technical problem instead of focusing on the customers' real needs. The decision of *how* to build, on the other hand, is still highly focused on a technology-first approach that limits beforehand the kinds of solutions that could be presented. And there's a good reason for that: the

inductive thinking, prevalent today in the industry, makes it impossible to see and understand the whole.

To illustrate, imagine one product team in a fintech company is tasked with building a debit card product from scratch to allow their customers to make payments from their accounts. They build an integration with the card network rails and the product is a great success. After a while, a second team is tasked with building a bill payment feature. They integrate with a bill payment provider and the product is a great success. Finally, a third team is tasked with building an account-to-account transfer feature and it makes a third integration to a transfer rail.

At the end of this process, there are three teams with three systems that are able to remove money from the customer's account and send it through a specific rail. The complexity can start crippling the teams' ability to move fast and add new features. The problem-first, systemic approach, on the other hand, could anticipate that debit cards are only one specific method of taking money from customers' accounts for specific purposes. In that case, the first team could have built an extendable system that, given a set of variables, could take a certain amount of money from the customer's account and send it with the appropriate metadata to its proper destination.

The kind of reasoning that is prevalent in the industry today is related to a cartesian way of thinking that still views the world as a machine comprised of smaller pieces that form a whole. That view, however, is part of an old paradigm in science that no longer holds true. From On Complexity (foreword), p. xxvi, xxvii:

"The key elements of the organization of knowledge in the West go far back in history. The work of Aristotle and Descartes is central. Aristotle developed a "logic," providing us with concepts such as the law of identity and the excluded middle. (...) Descartes was providing us with an orientation for the way we think, a focus on reduction, simplification, and clarity. *[Their contributions]* become the foundation for "good thinking," and *[were]* institutionalized in the organization of universities. There we find the same increasing specialization in departments, literally the splitting up into smallest possible parts, and the creation of strong boundaries based on three axioms of classical logic (Nicolescu, 2002).

The limitations of this kind of thinking are becoming increasingly apparent. None of the sciences offer us a way to integrate all the tremendous quantities of information and knowledge generated in the various disciplines and subdisciplines. This is extremely problematic for at least two reasons. First, with increasing specialization, the "big questions are simply not asked and addressed anymore. Second, action in the world cannot be confined to knowledge drawn from one discipline".

"As I have argued elsewhere (Montuori, 2005a) drawing on Morin's work, transdisciplinarity can be summarized as requiring: 1. A focus that is inquiry-driven rather than discipline-driven. This in no way involves a rejection of disciplinary knowledge, but the development

> of knowledge that is pertinent to the inquiry for the purposes of action in the world (...)".

We still live in the age of simplification. What philosophers call "paradigm of simplification" is the combination of the ideas of *disjunction*, *reduction*, and *abstraction* (On Complexity, p. 3). By *abstraction*, we mean that more concepts are abstracted away (aka ignored) by reducing the level of abstraction of the concepts people operate on. By *disjunction*, we mean the separation of the individual ("I think, therefore I am") and the object of study (the things I can think about). This separation gave birth to the segregation of the three big disciplines of thought: physics (the "exact" sciences), biology, and the humanities. To accommodate the complexity of reality into the ever-increasing specialized disciplines, *reductionism* took place. From On Complexity, p. 4, 6:

> "Hyper-specialization tore up and fragmented the complex fabric of reality, and led to the belief that the fragmentation inflicted on reality was reality in itself. (...) Blind intelligence destroys unities and totalities. It isolates all objects from their environment. It cannot conceive of the inseparable link between the observer and the observed. Key realities are disintegrated. They slip through the cracks between disciplines. (...) "We are blind to the problem of complexity. (...) This blindness is part of our barbarism. It makes us realize that in the world of ideas, we are still in an age of barbarism. We are still in the prehistory of the human mind. Only complex thought will allow us to civilize our knowledge".

That is a real problem for the tech industry. We need bridges in tech, not walls. By contrast with the hyper-specialization path, a systemic approach to technology would be domain-first and problem-driven. Let us explain.

By domain, we mean the context of the application of each piece of technology. That would mean perceiving domains of technology application as systems that must be understood in their elements and their relationships to one another. And the domains themselves as elements of greater systems, that interact with each other. Understanding the domain's lower elements (primitives) and how they interact with each other and can be combined allows us to build effective, lasting solutions.

This domain-first, problem-driven approach has similarities but is not exactly what Eric Evans proposes in the Domain-Driven Design methodology. From Microsoft's blog:

> "Good models exhibit a number of attributes independent of their implementation. The fact of the matter is, the mismatch between the model that's in everyone's head and the model you're committing to code is the first thing an aspiring domain modeler should understand.
>
> The software you create is not the true model. **It is only a manifestation**—a shadow, if you will—of the application Form you set out to achieve. Even though it's an imitation of the perfect solution, you can seek to bring that code closer to the true Form over time.
>
> In DDD, this notion is called model-driven design. Your understanding of the model is

> evolved in your code. Domain-driven designers would rather not bother with reams of documentation or heavy diagramming tools. They seek, instead, to imbue their sense of domain understanding directly into their code.
>
> The idea of the code capturing the model is core to DDD. By keeping your software focused on the problem at hand and constrained to solving that problem, you end up with software receptive to new insights and moments of enlightenment. I like what Eric Evans calls it: crunching knowledge into models. When you learn something important about the domain, you'll know right where to go".

DDD has the virtue of taking a step in the right direction and refusing the development of technology for technology's sake. By contrast, it focuses on the problem at hand and on knowledge beyond the engineering technique. However, from the conceptualization above, it should be apparent by now that Evans' approach has a strong cartesian logic to it. It suggests domains are boxes of context that can be statically defined and strictly delimited. That view, however, does not capture the complexity of the real world. When Evans say we should crunch knowledge into models, it misses the most important point: that these "models" that developers aspire to encode are themselves a manifestation of an underlying system where its domain rests. So to write code that is simple but not simplistic, one must go through the process of building models for the underlying systems they operate in and, in turn, write code that best represents such models. Otherwise, engineers will

be stuck writing code that represents loose, out-of-context knowledge they can't truly understand.

The result of this kind of reasoning, though, is that an unbelievable amount of ultra-specific and un-evolvable solutions are built and scraped every quarter. Talk about waste. It is the engineers' job to use code to abstract inherent complexity, not ignore it. By that, we mean to create software that can process complexity in a way that its users can have shortcuts to the endless possibilities the underlying domain may encompass. But if the software ignores the complexity, the result is a sterile solution with no means to integrate to the whole system and is destined to fail.

The mathematician and philosopher Blaise Pascal is believed to have fathered the original concept behind the saying, "If I had more time, I would have written a shorter letter". The core idea is that it usually takes longer to make something simpler. The same goes for software. The more abstract, simple, systemic, domain-first, problem-driven approach is one that takes more research and involvement from the engineer. After all, they need to figure out not only what is known about the problem, but also what are the underlying assumptions that can be drawn from the domain.

In the end, the approach of DDD connects to what many technologists think when they say overengineering should be avoided at all costs. The problem is that overengineering is a term that, by definition, means waste and inefficiency. No one would argue that overengineering software would be better than employing the right amount of engineering.

The issue is defining what that really means. Time and time again, software engineers end up advocating for ultra-specific solutions that solve one particular use case and cry "overengineering" to anyone willing to understand the underlying needs and context - exactly because it takes more time and greater expertise. The result is a simplistic model that escalates complexity at every turn instead of a model that is simple in essence and can evolve as new demands appear.

The Systemic Approach to Technology, on the other hand, considers all forces that influence the behavior of the agents within a system and their first and second-order effects. It takes considerations about policy, economy, and a number of different variables that may be relevant to model a comprehensive system of needs. For that reason, it's more abstract by nature and able to explain and sometimes predict market behaviors better than any other reasoning framework.

In the next topic, what a systemic, domain-first, problem-driven approach means in practice is discussed in greater detail.

c. An introduction to systems-driven design (SDD)

The previous topic discussed how the systems approach must be introduced in the decision about the scope of discovery and research into how software engineer teams should approach problems. The same systemic approach should also be applied to *how to build* technology - that is, how the solution should be designed. That means the

solution itself should be composed of elements that can be combined to solve an indefinite number of problems.

The systems-driven design is an approach to building technology solutions that involves identifying the primitives of a system and understanding how they interact with each other and with the larger system as a whole. By doing so, we can create flexible and adaptable solutions that can solve a wide range of related problems within a given system.

This approach allows for greater flexibility and adaptability in technology design, as well as a better understanding of the underlying logic and structure of a given system. It also encourages a systemic way of thinking about technology that takes into account the broader context and implications of a given solution, rather than simply focusing on the technical details.

Take Pix as an example. Pix is the real-time payments (RTP) rail built and run by the Brazilian Central Bank (BCB). It's the fastest-growing RTP rail in the world, settling over one thousand transactions per second at 99.99 availability rate (data available at BCB's website). The design of Pix is such that the rail is able to model any kind of payment solution known today while leaving room for untapped innovation.

The secret behind this kind of capability is systems-driven design. While designing Pix, the Brazilian Central Bank took a drastically different approach from the one prevailing at the time. It wasn't BCB's first time designing a payment rail. Brazil has many rails for multiple purposes. One for settling card transactions, one for payroll transfers, one for paper

checks, one for direct debit, another for bill payments, yet another for utility payments, and so on and so forth. What all these rails have in common is the fact they were built to solve one specific problem at a time. The consequence is a burst of accidental complexity, a spiderweb of intermediaries, high costs, and inefficiencies all around.

The new approach BCB took, on the other hand, was to build not a product but a platform upon which institutions could build any kind of payment solution without the need to create new integrations. The core concept that enabled this was the idea that all payment solutions, without exception, are all about moving money from one account to another, along with metadata about the payment. Anything, from credit cards to paying government taxes, follows this basic setting. Based on that realization, BCB knew it had to build a settlement chamber (clearing house) and a communication rail that could effectively connect any two members of the rail. After that, it only took designing a messaging protocol between participants that could be indefinitely extended to create the most flexible payment rail of all time.

II.3. Our approach to technology development must be propositive and focused on societies' greatest challenges

We're on the verge of the turn of a cycle, sociologically speaking. The opportunity to lead what's to be of the next one is here. So far, what has been getting attention are visions for the future constringed by decades-old lyrical dreams of science fiction. Virtual reality, self-driving cars, and AI as either messiah or exterminator: all conceived in

science fiction and brought to life by inspired technologists. Inspired by Mr. Morozov's clever wording, we can call them technological poets.

To be fair, there is nothing wrong with art inspiring science and conforming reality. In fact, there are times when turning to fiction is the best way of breaking off the constrictions of what is perceived as a possibility in science. However, humanity is now at a crossroads and it's time to press pause in this techno poetry, stop gazing at the stars, and start navigating toward our goals. We need skilled navigators willing to sail beyond the horizon line toward new discoveries. But we need to be clear about where we want to go and avoid the mistake of letting decades-old stories and start concentrating on what society needs today. We must learn to imagine new possibilities without the lenses of our old myths and intentionally stir the future of tech towards a society-focused approach.

When we say society-focused, it should be clear by now that we mean it in a systemic way. So what is good for society is not the same as what is good for individual consumers at any given time. It's the systemic harmonic equilibrium of forces and their first and second-order effects. It takes considerations about policy, economy, and a number of different variables that may be relevant to model a system of needs, as described in the previous topic about the Systemic Approach to Technology.

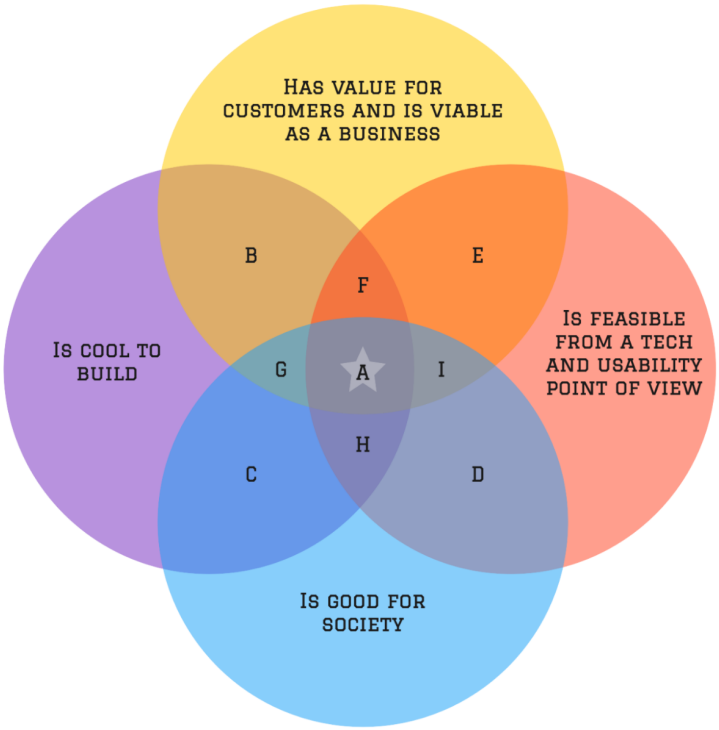
At this point, it's important to make something very clear. Our plea is not that law or regulation should mandate a society-focused approach. Nor is it a cry for boycotts or to *cancel* the techno-poets. It's about what we, as a society,

expect from tech from a propositive point of view. From a standard logic point of view, it might sound like it's the same thing to go against techno-poetry or support whatever is the opposite of that. However, from a practical, rhetorical, and behavioral point of view, to focus the debate and communications on what we *should* be doing is way more effective than calling out the opportunity cost of what we *could* do, but doesn't really advance our ultimate goals. The difference between these two approaches is that while the latter focuses on confrontation and blockage, the first puts forward a mindset for action and evolution.

Psychologically speaking, the things we pay the most attention to are usually the ones that grow the most. What is true for individuals is also true for societal groups, especially regarding the stock market. So, while we focus on the intersection between cool, technically feasible, and sellable, that's all the tech sector will ever do for us, with some outliers entering the realm of *advancing humanity's goals* in some criteria.

The diagram below illustrates the many areas where product ideas can find themselves. Each circle of the Venn Diagram designates an independent property a tech product may have:

Diagram of Values | Society-focused tech



F+E: current focus
F: technological poetry
E: boring but profitable business
H+D: NGOs, Governments arena
A+I: a new horizon to put our focus and energy
A: area where many companies retrofit to their goals but only few actually manage to occupy
I: boring but impactful and profitable business

One of the best minds in product management today, Marty Cagan, says product teams must build products that customers love and that are good for business (the yellow

and red circles in the diagram of values above). Both Cagan and Teresa Torres explicitly address the need for product teams to test the ethical question of whether certain products should be built.

That approach works perfectly for a company, but it still cannot be good enough for society. But we as a society are not thinking about what ***should and could*** be built, and yet is not currently being thought out. That should be the job of teachers, journalists, researchers, and every single person impacted by life in a globalized society. This is to say, these are our problems. Now, how can technology help, if at all?

In short, if the approach in business, as Marty Cagan summarizes, is: "to build something that customers love and is good for business," as a society, our approach should be to find and nurture those who can use the Systemic Approach to Technology to build something that society needs, customers love, and that is good for business.

As to the product teams inside startups and companies of all sizes and all industries, to have a society-focused approach means that society's interest should not be just a negative criterion, a reason to refrain from doing bad things, but a positive criterion, a reason to act and build good things. The society-focused technology, therefore, puts at the beginning of the product vision exercise the fundamental question of what society needs. Only then should we identify, within that scope, what are the products and services that customers want and companies can build and profit from.

III. Wrapping up

Our society-focused approach to technology is teleologic, systemic, and propositive. We believe that technology design and policy should be all about modeling the systems that will enable us, the tech industry, to better serve society; and that technology should be built from its primitives up in order to solve domain-level problems. By shifting our approach to technology, we can build solutions that are more effective, more sustainable, and more equitable.

Additional references:

- Fritjof Capra, The Turning Point, 1983.
- Edgar Morin, On Complexity, 2008.
- Clayton M. Christensen, Scott D. Anthony, Erik A. Roth, Seeing What's Next, 2004.
- Donella H. Meadows, Thinking in Systems, 2008.
- Karl Popper, The Logic of Scientific Discovery, 2002.

About the authors

Mariana Cunha e Melo is an experienced lawyer with a background in strategic litigation, public policy, legal research, and regulated markets. She has worked with Supreme Court justices in Brazil, represented Google in high courts and strategic litigation cases, and built the internet law team at a top law firm. At Nubank, she helped structure the Public Policy team and led efforts to work with the Brazilian Central Bank on designing its Real-Time Payments rail. She has also worked in early-stage startups as a strategic projects owner and director of regulation and strategy. Mariana is also a writer and speaker on topics such as privacy, free speech online, and regulation of fintech companies, with numerous international events under her belt.

About the authors

Jonas de Abreu is an experienced security engineer with a strong background in software engineering and security. He worked as an external consultant and instructor in the early stages of his career. Jonas spent almost a decade at Nubank, the leading fintech company in LATAM, where he made significant contributions as the Chief Information Security Officer and later as a Principal Engineer. He played a crucial role in shaping Nubank's infosec team and strategy and made a notable impact on the Brazilian Central Bank's decisions regarding the Pix rail. Jonas was responsible for Nubank's technical proposals during the Pix Forum, which were highly valued by industry experts and had a pivotal role in shaping the future of payment systems in Brazil.