CUTTER Business Technology Journal

Management, Innovation, Transformation

Vol. 31, No. 7, 2018

Architecture + AGILE

The Yin & Yang of Organizational Agility

Whynde Kuehn
Guest Editor

CUTTER Business Technology Journal

As business models for creating value continue to shift, new business strategies are constantly emerging and digital innovation has become an ongoing imperative. The monthly *Cutter Business Technology Journal* delivers a comprehensive treatment of these strategies to help your organization address and capitalize on the opportunities of this digital age.

Cutter Business Technology Journal is unlike academic journals. Each monthly issue, led by an expert Guest Editor, includes five to seven substantial articles, case studies, research findings, and/or experience-based opinion pieces that provide innovative ideas and solutions to the challenges business technology professionals face right now — and prepares them for those they might face tomorrow. *Cutter Business Technology Journal* doesn't water down or delay its content with lengthy peer reviews. Written by internationally known thought leaders, academics, and practitioners — you can be certain you're getting the uncensored perspectives of global experts.

You'll benefit from strategic insight on how the latest movements in digital innovation and transformation, IoT, big data analytics and cloud, to name a few, are changing the business landscape for both new and established organizations and how cutting-edge approaches in technology leadership, enterprise agility, software engineering, and business architecture can help your organization optimize its performance and transition to these new business models.

As a subscriber, you'll also receive the *Cutter Business Technology Advisor* — a weekly bulletin featuring industry updates delivered straight to your inbox. Armed with expert insight, data, and advice, you'll be able to leverage the latest business management thinking to achieve your organization's goals.

No other journal brings together so many thought leaders or lets them speak so bluntly — bringing you frank, honest accounts of what works, what doesn't, and why. Subscribers have even referred to *Cutter Business Technology Journal* as a consultancy in print and likened each month's issue to the impassioned discussions they participate in at the end of a day at a conference!

Get the best in thought leadership and keep pace with the technologies and business models that will give you a competitive edge — subscribe to *Cutter Business Technology Journal* today!

Founding Editor: Ed Yourdon
Publisher: Karen Fine Coburn
Group Publisher: Christine Generali
Managing Editor: Cindy Swain
Copy Editors: Jennifer Flaxman, Tara Meads
Production Editor: Linda Dias
Client Services: service@cutter.com

***Cutter Business Technology Journal®** is published 12 times a year by Cutter Information LLC, 37 Broadway, Suite 1, Arlington, MA 02474-5552, USA (Tel: +1 781 648 8700; Fax: +1 781 648 8707; Email: cbtjeditorial@cutter.com; Website: www.cutter.com; Twitter: @cuttertweets; Facebook: Cutter Consortium). ISSN: 2475-3718 (print); 2475-3742 (online).*

Subscription rates are US $485 a year in North America, US $585 elsewhere, payable to Cutter Information LLC. Reprints, bulk purchases, past issues, and multiple subscription and site license rates are available on request.

NOT FOR DISTRIBUTION
For authorized use, contact Cutter Consortium +1 781 648 8700 or service@cutter.com.

☐ Start my print subscription to *Cutter Business Technology Journal* ($485/year; US $585 outside North America).

Name _____ Title _____

Company Address _____

City _____ State/Province _____ ZIP/Postal Code _____

Email (Be sure to include for weekly *Cutter Business Technology Advisor*) _____

Fax to +1 781 648 8707, call +1 781 648 8700, or send email to service@cutter.com.
Mail to Cutter Consortium, 37 Broadway, Suite 1, Arlington, MA 02474-5552, USA.

Request Online License Subscription Rates

For subscription rates for online licenses, email or call: **sales@cutter.com** or **+1 781 648 8700.**

CUTTER CONSORTIUM
Access to the Experts

Opening Statement

by Whynde Kuehn

According to the principle of yin and yang, all things exist as inseparable and contradictory opposites. In this issue of *Cutter Business Technology Journal* (*CBTJ*), we explore the relationship between architecture and organizational agility as a powerful paradox: *architecture is the way to agility*.

Business agility — the ability of an organization to continually anticipate and react to change in response to major forces such as globalization and technology — is moving from the realm of competitive advantage to a necessity for survival. There have been great advances in improving the agility of execution, and while many organizations are pursuing such approaches, the bigger question remains: how does an organization truly transform to be agile at its core, including with end-to-end strategy execution and employee mindset? Organizations are also grappling with what the introduction of agile execution approaches means to longer-term, big-picture perspectives, such as strategy formulation and architecture. Driven by the realization that business agility is no longer optional for long-term survival, many organizations are looking to understand the depth of what agility really requires and the most effective path for achieving it.

Organizations are living organisms, constantly evolving to adapt to the environment. But today's environment is calling for a quicker sense of adaptation than ever before. We are shifting far more toward agile mindsets and ways of working. However, agility is not just for execution. Simply speeding up execution does not ensure we are doing the right things; in fact, we could be doing more of the wrong things faster! Quite the contrary, agility is a concept that ripples from strategy through architecture and then to execution.

The way organizations strategize, architect, and execute change needs to further evolve from linear and relatively static to continuous and flexible (see Figure 1). Can we reimagine strategy as an ongoing conversation constantly reacting to the environment, one that truly

	FROM	TO
Strategy	• Linear strategic processes	• Ongoing conversations for continual adaptation and innovation
	• Incrementalism and focus on planning and budgeting	• Focus on strategic choices and big moves
Architecture	• Role of "governor"	• Role of "enabler" for strategy and execution
	• Top-down architectures	• Flexible architectures designed for agility and resiliency; architectures "grown" not "designed"
Execution	• Strategy translation in silos, often producing redundant, misaligned, or nonintegrated solutions	• Architecture-based strategy translation to define a coordinated set of initiatives and solutions across organizational silos; flexibility for replanning
	• Focus on doing things quickly	• Focus on doing the *right* things quickly, with highly targeted resource utilization

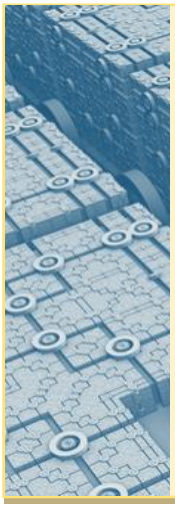
Figure 1 – Key shifts in strategy, architecture, and execution enable organizational agility.

focuses on strategic choices and big moves — versus an annual process that focuses on planning and budgeting for incremental changes, as is frequently practiced by many organizations today? And what if the results of those strategic conversations continually flowed into an architecture-based approach that translated them into a coordinated set of initiatives across organizational boundaries — instead of each organizational area interpreting the strategy and creating its own projects in silos, hoping that the results do not conflict and will add up to achieve the overarching strategy in the end? And could agile execution approaches be made more successful if they were informed by an architectural direction that defined an agreed-upon, rationalized set of business terms, big-picture business outcomes, and business priorities so that we could focus our precious resources with precision on doing the right things in the most effective way?

This issue of *CBTJ* demonstrates the seemingly contradictory idea that architecture — something often perceived as structural, static, constraining, governing — is actually the enabler for an organization to become more agile and fluid, from strategy through execution. Architecture is not a box to be checked. On the contrary, it is the mechanism for translating strategies into the right set of coordinated initiatives for execution. It is the *bridge* between strategy and execution, the bridge that supports the cross-organizational coordination and objective decision making that is so desperately needed in many organizations today.

The traceability provided through the architecture — from strategies and objectives down to the initiatives — also accelerates replanning as business direction shifts because it allows for the ready identification of impacts, enabling organizations to make decisions about what work to stop, pause, or continue. Target architectures can paint a picture of the future and intended business outcomes, helping to create context for people and allowing them to work toward a common vision, regardless of their role. At a foundational level, architecture can also be used as an enterprise lens with which to identify and simplify the business and technology environment in the first place, so that making changes can occur more quickly in the future.

Becoming an agile organization — and leveraging architecture as an enabler for various aspects of it — is a journey that requires deliberate actions and takes time. These changes must be underpinned by strong executive leadership and organizational change management to help shift peoples' mindsets to embrace continuous change and innovation. Moreover, it may require bold measures to support the change, such as realigning compensation and motivational mechanisms or adjusting investment processes to allow for initiative funding across organizational or portfolio boundaries. Indeed, organizations willing to invest today in reinventing their mindsets and mechanisms for continually anticipating and reacting to change will be the ones ready for tomorrow.



Upcoming Topics

The Critical Need for Data Governance
Claude Baudoin

Building a Digital Business Starts with a Data Warehouse
Barry Devlin

The Next Wave of Cloud Computing
Frank Khan Sullivan

In This Issue

In our first article, Cutter Consortium Fellow William Ulrich and I focus on business architecture and discuss how it can be leveraged as an enabler along the strategy realization path to harmonize the execution of business direction across organizational boundaries and initiatives. We find that organizations struggle with realizing strategies, particularly when those strategies cross business unit, product, and external business domain boundaries. Research shows that more often than not, failure to realize business strategies is not because a given strategy is ill-conceived, but more often due to the scope and impact of those strategies being vague or unknown, especially when they cross organizational boundaries and require collaboration. Using a case study, we highlight how business architecture can be used in strategy formulation, impact analysis, business design, program definition, and agile execution. Business architecture pinpoints the *right* things to do in the *right* place at the *right* time. In fact, we assert that "when business architecture is in place, adopted and leveraged ubiquitously, the gateway for an organization to transform itself into an agile enterprise is in place."

Next, Yesha Sivan and Raz Heiferman articulate concrete ways in which we can both shift our perspectives

and act to "agilify" our organizations. They describe how agile organizations think, what they can do, the abilities they must possess, and even the key technologies they should adopt. They also touch on the important interplay among leadership, culture, business architecture, and digital architecture. As Sivan and Heiferman wisely point out, "Gone are the days that an organization could plan for sustainable competitive advantage and build a five-year (or even three-year) strategic plan. The business environment has become chaotic, dynamic, and disruptive. Enter agility, as the new capability to develop transient competitive advantage with shorter planning and execution cycles. Welcome to the age of 'agilification.'"

Our remaining articles turn to the "Agile" (as in Agile software development) side of agility. Barry O'Reilly and Gar Mac Críosta begin by pointing out how the worlds of Agile and architecture can't quite fit together. To resolve this, they introduce a new architectural approach, asserting that by "*architecting for antifragility*, businesses can gain real agility and deliver systems with a higher level of quality." They describe the challenges of complex systems and then define an Antifragile Systems Design process, which embraces the complexity in building dynamic systems to guide architects to optimize and balance volatility, uncertainty, complexity, and ambiguity on any project. The result of this approach is "a business with a better understanding of its own fragility and a software system capable of bending and meeting the needs of the changing business environment."

Jan-Willem Sieben, Jan-Paul Fillié, and Cristina Popescu explore Agile architecture and architectural agility and how these two fundamentally different paradigms can reinforce one another. They describe the pitfalls or "anti-patterns" for both enterprise architecture and Agile — and then make a case for how they can be overcome by combining the practices. Based on real-world case studies from several organizations, Sieben, Fillié, and Popescu describe several specific ways in which organizations have combined architecture with Agile thinking and methods to break through the anti-patterns and improve their results.

Next, Bob Galen lays out nine rules of Agile architecture to inform us in how to think about architecture and help us strike the right balance between architecture and agility. He asserts that "software architecture requires balance. Often, you can focus too much on it, creating robust products that miss customer needs or over-engineer solutions. Conversely, especially in Agile contexts, you can under-engineer things and your product efforts can succumb to relentless refactoring rework. So there's a balance to strike in architecture, no matter what methodology you use to create your software. In Agile contexts, that balance is often lost." As Galen asserts, this balance comes at the point where you "define, refine, and implement just-enough and just-in-time architecture." The nine rules are indeed good ones to embrace — practical, grounded in experience, encompassing of the human aspects, and even delivered with levity.

To conclude our issue, Miklós Jánoska provides a perspective on how we can shift architecture from a governor to more of an enabler. As he points out, "Despite the multitude of architecture frameworks and methods, experiencing a smoothly working, pragmatic synergy between delivery teams and the architecture discipline is rare." He provides insight into common difficulties in Agile projects and then describes how organizations can establish a "non-blocking" architecture governance practice for Agile development teams. The approach leads to organically integrating architecture into the process, into the delivery pipeline, and into the teams' everyday work. As Jánoska reminds us, "Under the increasing pressure of accelerating marketplaces and rapidly evolving technologies, internal velocity and responsivity become significant differentiating factors. Indeed, responsive, flexible software ecosystems enable high-speed businesses."

No matter where you are on your journey toward organizational agility, we hope that the articles in this issue give you a new perspective on the art of the possible and on how architecture can be the gateway to a future of continuous adaptation, innovation, and success for your organization.

Whynde Kuehn is a Senior Consultant with Cutter Consortium's Business & Enterprise Architecture practice and Principal of S2E Consulting Inc. She is a long-time business architecture practitioner, educator, and industry thought leader who takes a business-focused and results-oriented approach to business architecture. Ms. Kuehn has extensive experience in enterprise transformation and planning and has a track record of creating successful teams that become embedded into their organizations. Ms. Kuehn also provides business architecture training. She has developed and taught comprehensive business architecture training programs via in-person and online formats, both for the public and inhouse for clients. She is a recognized thought leader in business architecture, regularly speaking, writing, and chairing/cochairing conferences and events that advance best practices and facilitate community across the world. She is a cofounder and board member of the Business Architecture Guild and serves as its Editorial Board Chair. Ms. Kuehn also founded a New York Business Architecture Community. She can be reached at wkuehn@cutter.com.

ADDING IT ALL UP

Business Architecture + Agile = Doing the Right Things, Fast

by Whynde Kuehn and William Ulrich

Organizations struggle with realizing business strategies, particularly when those strategies cross business unit, product, and external business domain boundaries. The question is: why? Research shows that more often than not, failure to realize business strategies is not because a given strategy is ill-conceived but more often due to the scope and impact of those strategies being vague or unknown. This issue becomes magnified in scenarios where the impacts of a strategic directive extend beyond a planning team's line of sight and require cross-business coordination and collaboration. In other words, the inability to realize business strategies is oftentimes the result of doing the wrong things in the right places and the right things in the wrong places, turning good ideas into failed projects, lost opportunities, and wasted investments.

Consider an example from a US federal government agency that highlights the need to hone in on strategic objectives, programs, and related investments on clearly defined aspects of the business from the start. An intellectual property (IP) office sought to transform cross-agency and public engagement by creating a highly transparent business ecosystem across the agency, along with its partners and constituents. The program scaled up to over a dozen projects across multiple business units, teams, and technologies, with each team centered on a particular objective and scope of impact. One team, in particular, worked toward transforming the processing of international registrations across the agency, including engagement with a third-party global organization.

Program executives assigned the end-to-end transformation of the international registration value stream to a team of business analysts and solution delivery personnel. While program leadership designated the scope of the effort to a specific value stream, enabling capabilities, and stakeholders, the analysts ignored the directive in favor of a "blank page" approach. Analysts met with business subject matter experts and legacy analysts, the latter of which directed the team to concentrate on constituent petitions as a priority, with the source of this diversionary thinking resulting from constraints of the current environment. International registration transformation was, in fact, supposed to eliminate the need for costly, time-consuming constituent petitions, except where essential.

A routine review by program management discovered that the project team had invested nearly three months of time on petition processing, while ignoring its primary focus on international registration transformation. Indeed, another project team had mobilized to work on end-to-end petition processing transformation, framed by an entirely different value stream. In short, the blank-page approach taken by the analyst team resulted in lost time, wasted investment, and, worst of all, alienation of business stakeholders.

What does this story have to do with the use of business architecture to establish an agile organization? Simply put, the false start exemplified by the wayward project team in this story repeats itself dozens or even hundreds of times at medium-to-large enterprises annually, resulting in an endless spiral of ineffective strategy realization and lost opportunities. Business architecture changes the game by enabling organizational agility through effective, coordinated translation of business directives into targeted results from strategy formulation through strategy realization. It allows organizations to frame the scope of business strategies, programs and projects, and related investments from the start in clear, unambiguous terms. Indeed, business architecture lays the foundation for expediting strategy formulation through strategy realization, ensuring that business investments center on doing the *right* things in the *right* place at the *right* time.

The Path to Successful Strategy Realization

Business architecture helps organizations shift toward becoming more agile from multiple perspectives. It plays a vital role throughout the strategy realization

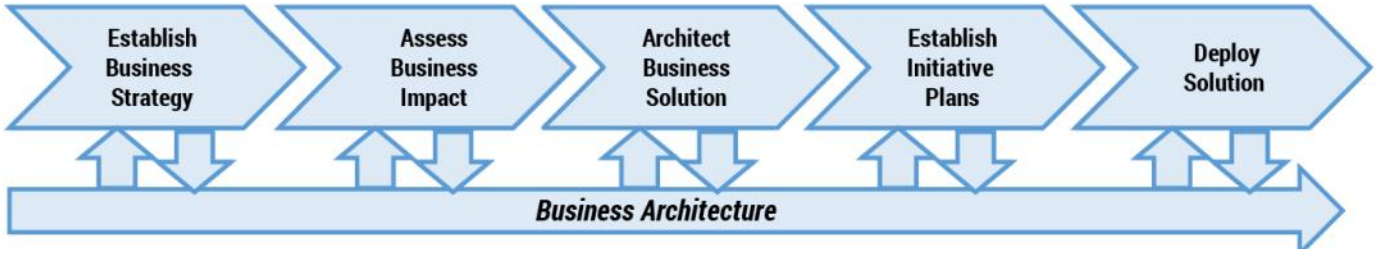


Figure 1 — An enterprise perspective on strategy realization. (Source: Business Architecture Guild®.)

path to increase the speed and effectiveness with which strategies become translated into initiatives, and with which initiatives introduce working changes to the business environment. Figure 1[1] shows an enterprise perspective on strategy realization for an organization, which occurs continually as a business implements strategies, transformations, and related business directives. End-to-end strategy realization requires many people to work together seamlessly across five stages; this includes teams centered on strategy, customer experience, architecture, product management portfolio management, program and project planning, business analysis, business process, organizational design, and execution. Business architecture is a relatively new addition to the ecosystem of strategy realization, but has a valuable role in all five stages, especially, but not solely, in Stages 2 and 3 (Assess Business Impact and Architect Business Solution).

These two stages are often skipped when organizations jump from establishing business strategy to executing projects in order to "execute faster." However, what may appear to be a decision to expedite delivery timelines actually requires an even larger investment of time, and often budget, later, when extra effort is needed to get the organization back on track after implementing a misaligned solution, or when it takes additional time to make a new change because redundant solutions add to overall environmental complexity. The intangible impact of these decisions can also include customer experience issues, business stakeholder fatigue, and even regulatory or reputational risks.

In the first stage, Establish Business Strategy, business architecture can inform strategy formulation, identifying new options for business model evolution. This stage can also help identify potential impacts of strategic options as they are being defined in order to quickly narrow down to viable options, resulting in significant time and mindshare saved in the long run.

In the next stage, Assess Business Impact, business architecture offers an invaluable enterprise-level business lens that allows strategy impacts to be comprehensively assessed for the entire organization, across all business units and products, internally and externally. Indeed, Stage 2 exposes the "butterfly effect" of all value stream and capability changes necessary to enable a given strategy, fanning out to highlight the effects on other strategies, stakeholders, new or existing products, policies, current or planned initiatives, processes, and, assuming there is alignment with the IT architecture, technologies.

Having a high degree of transparency of the full scope of the enterprise empowers teams to work together in new ways across organizational boundaries, versus each team translating the strategy in isolation and implementing its own projects. This leads to the third stage, Architect Business Solution. Here, business and architecture teams work together (across organizational boundaries, where necessary) to design changes to the business and technology environment or to design new business solutions for any affected components, such as value streams, capabilities, products, and system applications. This perspective represents a significant shift in thinking. It is an important step forward for creating an agile organization because it ensures that the right solutions are built based directly on the business strategy and built only once in an integrated manner.

For example, the IP organization mentioned earlier was able to bring various people together from across the agency as well as its partners and constituents to comprehensively design the international registration value stream and enabling capabilities versus multiple departments and stakeholders trying to solve the problem in their own silos. In this scenario, capabilities were automated once and reused over and over again across various projects and solutions as required by a given strategic directive. One last point regarding Stage 3: this is the point where organizations can truly engage in design thinking based on a well-defined strategy and highly transparent impact points. The

degree of transparency provided by business architecture lays the fertile groundwork for exploring a wide range of stories and what-if scenarios across the business ecosystem.

Once a business solution has been designed, Stage 4, Establish Initiative Plans, leverages business architecture to determine how to best allocate the full scope of work across initiatives, which often break down into multiple programs and projects with clearly defined scope and delivery sequence. When organizations skip Stages 2 and 3, the resulting projects are often overlapping, sometimes even contradicting each other, and may not be sequenced effectively. Often, the scope of a given program or project is too large or too small, but implementation teams discover this too late in the cycle to convince management to retrench. This situation can even happen when people actively collaborate across initiatives because they are limited by a fragmented, opaque understanding of the bigger picture and organizational constraints.

Expediting strategy definition through strategy realization requires a clearly defined, end-to-end business focus that articulates the specific investment targets associated with a given business objective.

At the beginning of Stage 4, the scope of each initiative is framed by concrete changes to the architecture, with a clear articulation of what needs to change or be created using the common architectural components (i.e., value streams, capabilities, and applications). This activity requires business, architecture, and planning teams to work closely together, and again represents a significant shift in thinking related to how and when initiatives are defined. It ensures that initiatives are scoped in the right way and delivered at the right time. Our IP case study and the wayward project team that centered on petitions instead of the defined architectural focal point for the initiative provide an example of what can happen when this top-down approach is not used (or is disregarded): countless weeks, months, or years of precious time and resources are wasted with zero business value delivered.

Finally, in Stage 5, Deploy Solution, each initiative goes through its usual cycle of execution. Regardless of development method used, waterfall or Agile, the business architecture ensures that project teams have the right focus at the right time. Business architecture also provides a reusable framework for defining, tracking, and aligning business requirements, user stories, and deployed software solutions. The next section articulates in detail the value that business architecture can bring to agile execution approaches.

The Value of Business Architecture in Agile Execution Approaches

Expediting strategy definition through strategy realization requires a clearly defined, end-to-end business focus that articulates the specific investment targets associated with a given business objective. Figure 2 highlights this perspective. A series of objectives shown to the left are handed over from business executives. Planning teams associate the objective with a given value proposition (in the IP example, obtaining an international registration), which then becomes the focal point of the overall planning effort and investment.

Figure 2 depicts the value streams as the focal point of the analysis. Value streams are associated with external and internal stakeholders and enabling capabilities, which in turn are associated with business units, third parties, information used by enabling capabilities, policies linked to capabilities, and the technologies that automate the enabling capabilities. Through these connections, the business ecosystem impacted by the originating strategy comes into clear focus, which in turn frames the investment targets and program scope.

In the IP investment example, the agency sought to ensure total transparency of the international registration value proposition while expediting delivery of the end-value proposition. In this example, that meant reducing the need to file a petition unless absolutely necessary. The business objective pointed to the value proposition of an international registration, covering every applicable international jurisdiction. The value proposition pointed to the registration value stream, which engaged a cross-section of external and internal stakeholders, including the applicant; enabling capabilities; business units, including partners; and related information.

The resulting project was entirely framed around this value stream and related business perspectives. A

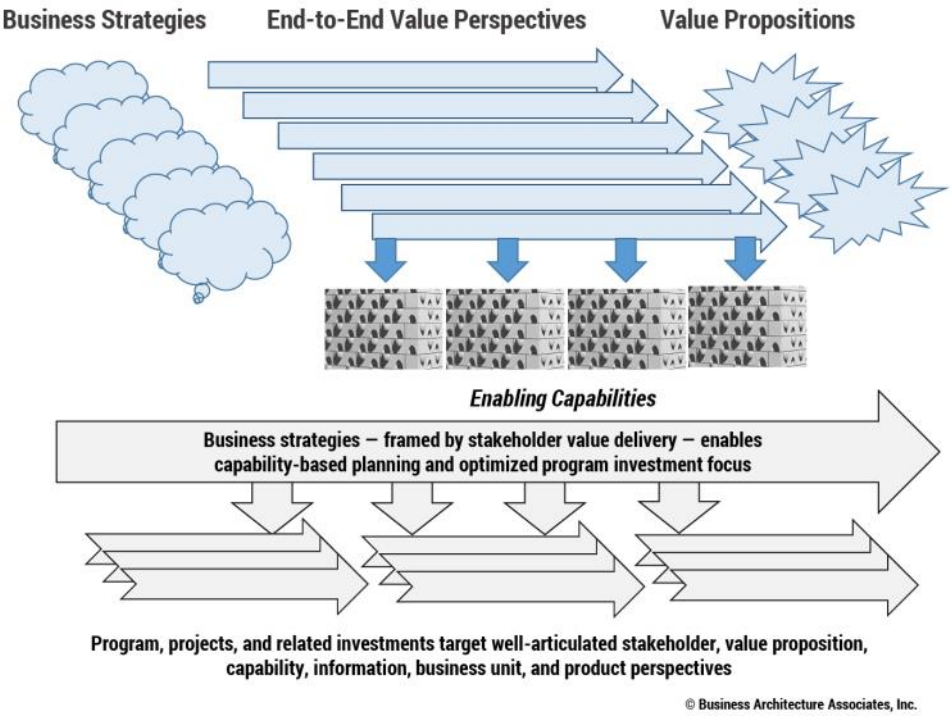


Figure 2 — Targeting business investments through business architecture.

second parallel project and team directed its energy and efforts on the value stream to decide a petition, thereby avoiding conflicts or overlap of the work being done and results being achieved. The intentional points of overlap involved shared enabling capabilities across the value streams and projects, which highlights a key advantage in applying the business architecture approach: shared enabling capabilities point to opportunities to establish reusable software deployments. Each project and project phase deployed or enhanced capability-aligned software services that became reusable in later project phases, allowing the program to scale to concurrent projects while reducing delivery timelines.

Business Architecture Provides Common Vocabulary and Mental Model

In his 1833 book *On War*, Carl von Clausewitz states that "the first task of any theory is to clarify terms and concepts that are confused…. Only after agreement has been reached regarding terms and concepts can we hope to consider the issues easily and clearly, and expect others to share the same viewpoint."[2] Expediting strategy definition to strategy realization also requires a common vocabulary to be leveraged across all business units or even external stakeholders who may be involved in developing solutions and helping the organization achieve its goals. This vocabulary must include a rationalized view of business terms like customer, product, and agreement for most organizations as well as any industry- or organizational-specific terms. For example, the IP organization defined common terms such as, for example, intellectual property, policy, research, classification, proceeding, and petition, as part of its business architecture.

Rationalizing and defining the business vocabulary is one of the most introspective and challenging things an organization can do. However, the shared vocabulary and mental model of an organization, which business architecture establishes through capability, value stream, and information mapping at the core, save immeasurable time that is otherwise lost in conversations where people talk over each other or solutions are developed that do not meet business needs. Fortunately, business architecture reference models have evolved to the point where the time and effort it takes to establish a business architecture baseline is dramatically streamlined. Moreover, a rationalized view of an enterprise is the foundation for any organization that plans to shift toward becoming a cognitive enterprise, where the cognitive enterprise is a learning organization with a centralized knowledgebase that evolves and accrues business intelligence over time.

Business Architecture Fosters Strategy/ Initiative Alignment and Prioritization

Consider that a given organization will often have many strategies it wishes to address. In the IP example, the executive team had a long list of strategic objectives that it had to prioritize across a five-year program and dozens of projects and project teams. The two examples discussed thus far involved one project designed to enable expedited, effective international IP registration with a second project centered on streamlining effective petition resolution. To accommodate executive directives and priorities, the agency had to execute several parallel projects under a single program, as illustrated in Figure 3.

In the IP transformation scenario, the overall program ensured that projects moved very quickly from stated objective to the delivery of work, which applied Agile project delivery principles leveraging two-week sprints, Agile epics and user stories, daily standups, backlog management, and Scrum-of-Scrums. With one exception of the case where the initial international registration project team went rogue and refused to leverage the value-stream-and-capability-framed project perspective, work progressed efficiently and smoothly from value stream stage to stage, across targeted stakeholders, focused on automating or otherwise improving enabling capabilities at each phase.

The example of the project team that went rogue makes a strong case for using business architecture to help frame business strategy realization. While project teams leveraged business architecture–framed priorities to expedite startup time, streamline business discussions, and reduce exploratory scoping, the rogue team wandered lost for three months, working on the entirely wrong business focal point and costing the organization time, money, and business credibility. In other words, teams that leveraged the business architecture moved with agile aplomb while the rogue team struggled to get its footing, demonstrating how business architecture makes a real difference between the expedited, successful delivery of business value and a series of failed investments and commitments.

Business Architecture Scopes Initiatives and Provides Deployment Framework

In the IP example, the overall program framed a collective set of strategies to be delivered in coordinated fashion, where each project targeted a given value stream or subset of a value stream, and work was prioritized by stakeholder and capabilities for each value stream stage. Consider the specific prioritization and sequencing of work associated with the IP registration value stream. Projects decomposed into four phases, each of which delivered quarterly. Each phase decomposed into a series of two-week sprints. Each sprint targeted delivery of certain capabilities for a given stakeholder, in the context of the value stream stage framing that piece of work. Figure 4 depicts the program, project, phase, and sprint decomposition concept.

In the IP case, the IP examiner was the initially prioritized targeted stakeholder within a value stream stage of the registration value stream. Project teams delivered capability-related enhancements and automations for the IP examiner over a period of multiple project phases, until there was a deployable solution for the attorneys to perform their jobs. The project prioritized additional stakeholders across business units, sprint

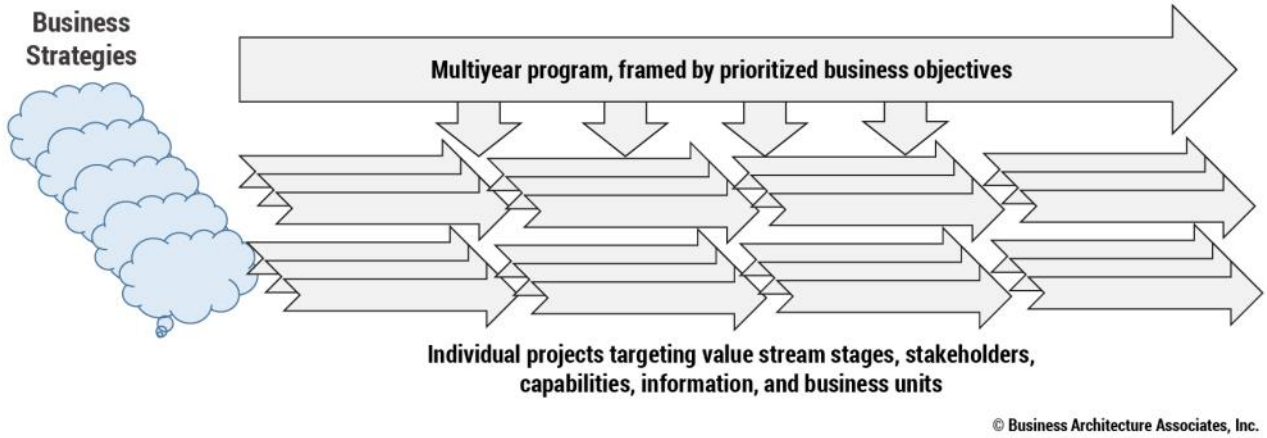


Figure 3 — Strategies delivered through clearly delineated, business architecture–framed projects.

after sprint, phase after phase, until the solution extended across all stakeholders for essential capabilities across that value stream.

This overall framing of programs, projects, phases, and sprints highlights the value business architecture brings to an organization using Agile delivery approaches. The IP program was employing Agile development techniques in both the challenged project associated with the rogue team and each successful project. When the rogue team on the international registration project was replaced with a new team that embraced and leveraged the business architecture, that project quickly regained its footing and began to deliver value across the stakeholder ecosystem. The organization achieved its overall strategic objective for the applicant as well as the internal stakeholders via project evolution and solution deployment. In other words, business architecture clearly made the difference between successful deployment of this project versus a challenged attempt at this project.

Business Architecture Connects the Dots Across Strategies, Architectures, and Initiatives

Business architecture is not only valuable within the context of one program; it also has the unique ability to connect dots across programs, which can be challenging in its absence. The shared enterprise-level business lens of business architecture, with a focus on value streams and capabilities, becomes the common key for cataloging changes being made as a result of any strategy, target architecture, or initiative. This formal framing of strategy realization makes it possible to definitively identify when changes are being made to the same area of the business, thereby uncovering opportunities for new collaborations, shared solutions, and coordinated decision making, such as when to pull back when too much change is being introduced for a given stakeholder.

The IP example highlights additional points in terms of connecting the dots from a holistic perspective. Having enterprise-wide visibility into all strategies and potential changes centered on international registration gave the IP program team a quick way to pinpoint additional factors that should have been considered up front, such as new treaties and regulations. Another factor was that the international work was eased by deploying a domestic IP registration solution first, which established a reusable baseline for the international solution. The program also had end-to-end visibility and impacts from a global software solution perspective. In this case, a shared database and reusable software services library emerged as a result of this overall program, meaning that the agility gained during the deployment stages of the program would be formalized and institutionalized to enable this organization to become and remain an agile enterprise well into the future.

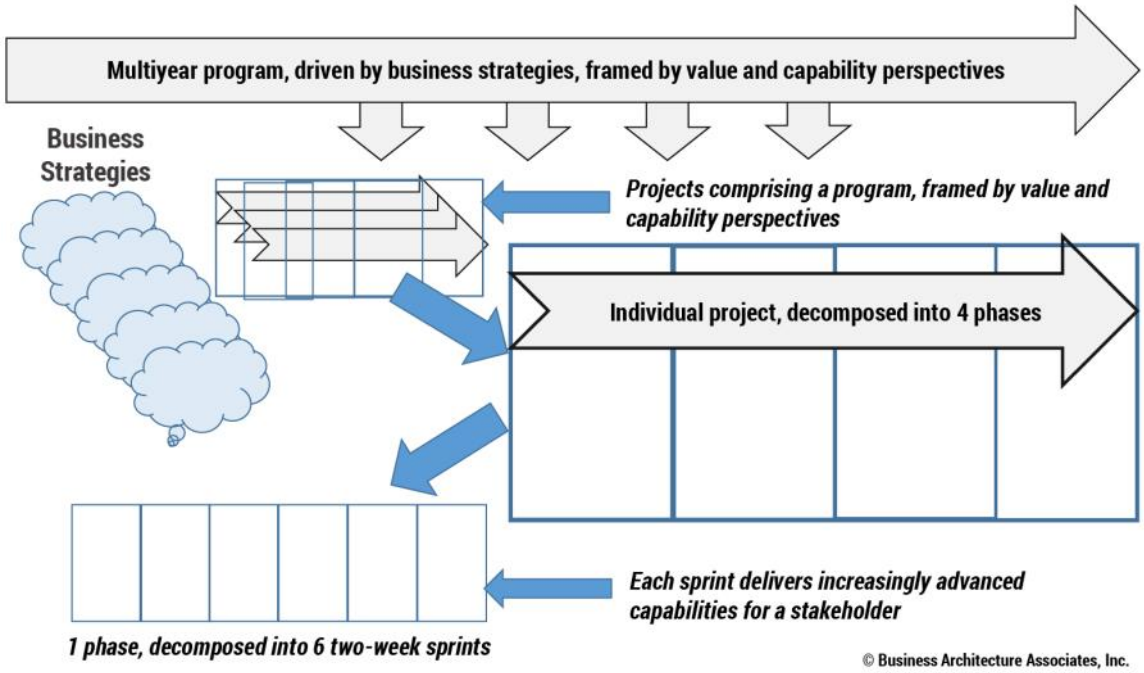


Figure 4 — Program decomposed into projects, phases, and two-week sprints.

Ubiquitous Business Architecture

One major differentiator of business architecture is that it is not constrained to internal business perspectives. The international value stream in the IP example highlights an external perspective. The original perspective of the business on international registrations essentially ended when the registration request was sent to the third-party global organization, but the value proposition was nowhere close to being achieved. Extending the applicant value delivery perspective end to end expanded and clarified the business objective. Stakeholder transparency had been constrained to internal views and lacked perspective on the third-party global organization engagement. This represented a design thinking phase of the program where no one ever conceived of establishing 360-degree visibility into work once it entered the partner domain. Yet the value stream made this external perspective blindingly apparent.

> *The business architecture should be created from an enterprise perspective and based on a rationalized view of business terms.*

The fully expanded stakeholder perspective highlights the concept of ubiquitous business architecture where value streams are not constrained to a single organization but rather across a business ecosystem that includes other business entities. By expanding the value stream perspective through delivery of all international registrations, the agency delivered major improvements in applicant value delivery. In addition, by adding capabilities to enable application engagement across the value stream, the need to initiate related petitions was reduced dramatically, which further reduced the demands on an already overworked business unit.

In order for an organization to maximize the value of business architecture, it should leverage a holistic ecosystem perspective on business architecture. This approach focuses on stakeholder value delivery from an end-to-end perspective, which frames the value streams targeted for investment. When engaged in this approach, the basic foundation for an organization to transform itself into an agile enterprise is in place.

Introducing Business Architecture into Agile Approaches

How then do organizations begin introducing business architecture into agile execution approaches to achieve these benefits? Those organizations that already have a business architecture in place have an advantage in that they can immediately begin leveraging it to prioritize and structure an agile execution. However, this requires a relatively complete and well-structured business architecture. For example, the minimum foundation includes the definition of core value streams and capabilities decomposed down three levels, as well as value stream/capability cross-mapping for areas of the business being targeted by a given set of strategies. Another key aspect of business architecture that comes into play early in the cycle is the information, which formalizes the business vocabulary and relationships across various information concepts.

The business architecture should be created from an enterprise perspective and based on a rationalized view of business terms. From here, the business architecture may be cross-mapped to other business architecture domains, such as strategies and products, as well as to other disciplines, such as event modeling, processes, and software applications. For those organizations that do not yet have a business architecture, it can be built over time once the value stream and capability foundation is in place. Reference models expedite this effort. From there, refinements, additions, and cross-mappings to the business architecture are captured opportunistically as dictated by business priorities.

In addition, the use of business architecture must be integrated into the way people work during an agile execution. For example, product owners need to learn how to work with business architects and use the business architecture to inform prioritization, and all project teams need to learn how to consume architectural scope and input for their projects and sprints. Business analysts must become fluent in understanding and using the business architecture. Adoption in practice can be more challenging than creating the business architecture in the first place because it requires the patience and desire to introduce a new frame of reference into one's mindset, albeit one that provides significant business transparency.

Making these types of changes is, of course, more successful when supported by other overarching measures to encourage and support people throughout the organization to shift to an agile and enterprise-focused mindset. This can include actions such as executive messaging to describe how the organization of the future will work and deliberate change management activities to help people adjust. Further measures may also be needed, such as adjusting funding mechanisms to work across organizational boundaries or adjusting employee compensation and motivational structures.

Conclusion

Organizations should leverage business architecture throughout the path to strategy realization to harmonize the execution of business direction across organizational boundaries and initiatives. Business architecture ensures that organizations do the right things that align with business priorities and are scoped and integrated effectively. Though business architecture is a relatively new addition to the ecosystem of strategy realization teams, it plays a valuable role in strategy formulation, impact analysis, business design, program definition, and agile execution. When business architecture is in place, adopted and leveraged ubiquitously, the gateway for an organization to transform itself into an agile enterprise is in place.

Endnotes

[1] *A Guide to the Business Architecture Body of Knowledge® (BIZBOK® Guide), Part 1*. Business Architecture Guild (https://www.businessarchitectureguild.org/page/about).

[2] "On War." Wikipedia (https://en.wikipedia.org/wiki/On_War).

Whynde Kuehn is a Senior Consultant with Cutter Consortium's Business & Enterprise Architecture practice and Principal of S2E Consulting Inc. She is a long-time business architecture practitioner, educator, and industry thought leader who takes a business-focused and results-oriented approach to business architecture. Ms. Kuehn has extensive experience in enterprise transformation and planning and has a track record of creating successful teams that become embedded into their organizations. Ms. Kuehn also provides business architecture training. She has developed and taught comprehensive business architecture training programs via in-person and online formats, both for the public and inhouse for clients. She is a recognized thought leader in business architecture, regularly speaking, writing, and chairing/cochairing conferences and events that advance best practices and facilitate community across the world. She is a cofounder and board member of the Business Architecture Guild and serves as its Editorial Board Chair. Ms. Kuehn also founded a New York Business Architecture Community. She can be reached at wkuehn@cutter.com.

William M. Ulrich is a Fellow of Cutter Consortium's Business & Enterprise Architecture practice and President of TSG, Inc. Specializing in business and IT planning and transformation strategies, he has more than 35 years' experience in the business-IT management consulting field. Mr. Ulrich serves as strategic advisor and mentor on business-IT transformation initiatives and also serves as a workshop leader to businesses on a wide range of business-IT transformation topics. He has the unique ability to cross business and IT boundaries to facilitate and streamline business-IT transformation strategies, and his workshops on business-IT architecture transformation have been widely attended by organizations worldwide. Mr. Ulrich is the cofounder and President of the Business Architecture Guild, Cochair of the OMG Architecture-Driven Modernization Task Force, and cofounder and Partner at Business Architecture Associates, Inc. Previously, he served in a senior management capacity at KPMG, including as Director of Reengineering Strategies, prior to leaving and forming his own company. Mr. Ulrich is a Certified Business Architect® (CBA®) and continues to play a role in formalizing industry standards around the practice. He has authored hundreds of articles appearing in major publications and journals and is coauthor of Business Architecture: The Art and Practice of Business Transformation. *He can be reached at wulrich@cutter.com.*

READY, SET, GO!

Agilifying Your Digital Organization: 6 Steps to Get Started

by Yesha Sivan and Raz Heiferman

> The only sustainable advantage you can have over others is agility; that's it. Because nothing else is sustainable, everything you create, somebody else will replicate.
>
> — Jeff Bezos, Amazon founder

In this article, we share six recommendations for those working in established organizations who have received a call from the CEO or upper management requesting something like, "Please make us agile."

The title of this article hints at our initial assumptions:

- **Agilifying.** Agility is not a specific goal. Each organization will have its own flavor of agility, depending on its history, legacy, "shareholder" goals, market and ecosystems, changing conditions, culture, and the like. Agility is a muscle that can be trained and enhanced. Our focus in this article is on the actions that will enhance overall organizational agility.
- **Your.** Any organizational endeavor needs an owner. Becoming agile means initiating, leading, and coordinating diverse efforts within the organization as well as solving conflicts that stem from these actions. The recommendations we include in this article are directed at the leader.
- **Digital.** We use the generic term "digital" to denote the unique place of information systems and other digital technologies in enabling agile. Today's organizations are structured around digital processes. Digital is the nervous system that allows the organization to plan, implement, and evaluate its endeavors, such as enhancing the customer experience and/or promoting new, innovative business models.
- **Organization.** Being agile is for every organization: small, medium, and large — whether you are a five-person startup or a 50,000-person conglomerate; local or global; private or public; for profit, nonprofit, or government. You can even build agility into your department, your team, and yourself. While the language of this article talks about the organization, the recommendations are relevant, mutatis mutandis, to all levels.

Gone are the days that an organization could plan for sustainable competitive advantage and build a five-year (or even three-year) strategic plan. The business environment has become ever-more chaotic, dynamic, and disruptive. Enter agility, as the new capability to develop transient competitive advantage with shorter planning and execution cycles. Welcome to the age of "agilification."

Step 1: Appreciate the Mental Challenge of Agility as "Building Flexible Buildings"

There is a giant universal mental challenge when it comes to agilifying. Think about standard buildings and the process of creating them. Thousands of years of experiences, images, mental models, tools (digital and otherwise), sample contracts, professional backgrounds, past knowledge, and people's expectations — all stem from the assumption that you architect and build in a linear way. You analyze the need, you make the plans, you build the building, and even though you may add just a few more things in the future, the building is a fixed entity. Any attempt to design a building for change — let alone *many* changes — will not be appreciated, to say the least.

Yet the goal of the modern organization is to rebuild itself all the time. To arrive at various and different structures, processes, and business models. To match the needs of the market. Let's envision our organizational building à la photographer Victor Enrich, who used Adobe Photoshop to demonstrate various future possibilities, including those shown in Figure 1.

The common response to the idea of flexible buildings is lukewarm, ranging from "you are crazy," to "nice idea but not practical," or, at best, "nice idea, now let's talk about something else." As an agilification leader, you will receive similar responses inside established organizations. The responses may not be as blunt, but the innate mental challenge is a formidable barrier. Years and years of organizational culture are geared to fixed systems, not agile ones.

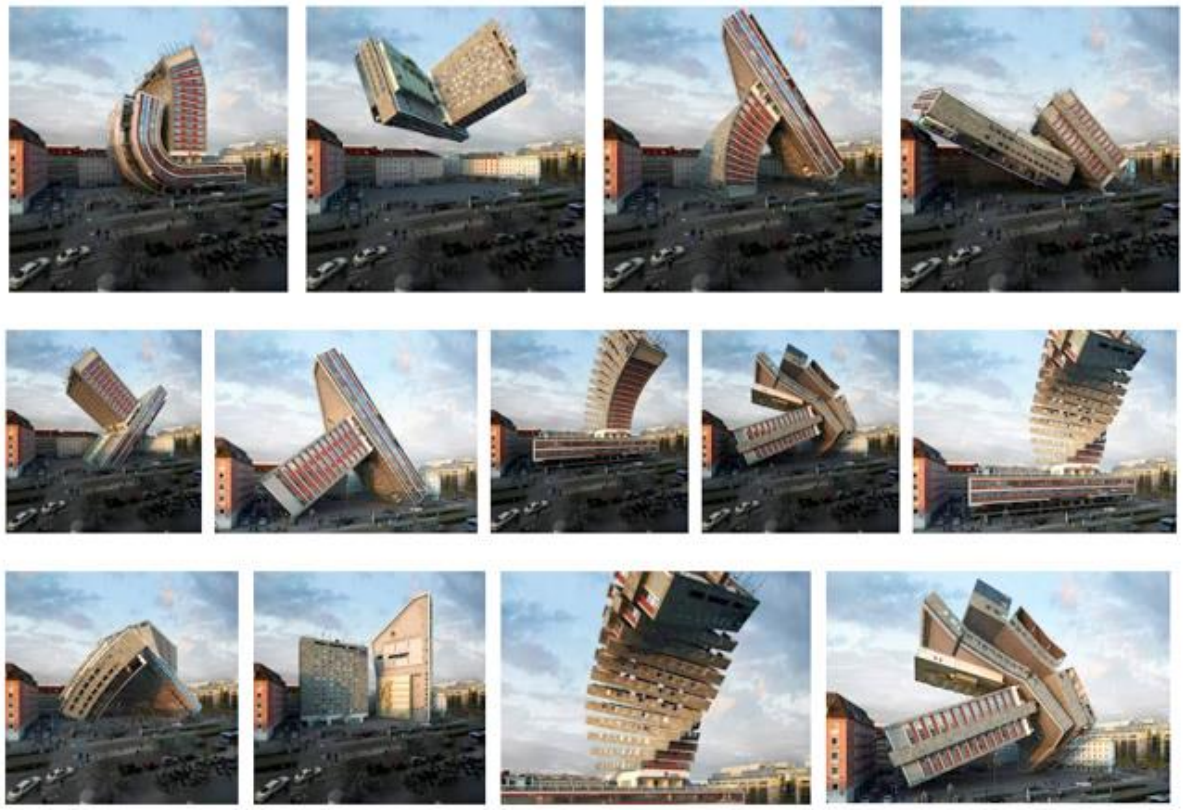


Figure 1 – Think "flexible buildings," not "fixed buildings." (Source: "NHDK" series by Victor Enrich, Munich 2012.)

Step 2: Develop and Share Visions of What to Expect from Agility

A good way to combat the innate mental objection to agility is to start from what can and should be done with agility, and how the organization will look and feel when it is agile.

Agility can mean different things to different people, and it should. Still, it is important to vividly understand its various meanings and to allow the organization to be aware of these meanings and then prioritize them. For example, an agile organization will be able to do the following:

- Release features faster (from once a year to once a quarter, then every month, then every day).
- Digest new product lines faster (e.g., as a result of M&As).
- Divest a product, department, or region (including all relevant IT abilities).
- Integrate a new small-to-medium-sized firm (through M&A), keeping the good DNA of both organizations.
- Divest a current small-to-medium-sized department.
- Change business processes fast and replicate them quickly to the entire global organization.
- Introduce new and innovative business models.
- Make decisions based on data, including decisions around experiments, insights, and corrective courses.
- Retire and replace old legacy systems (with minimal effect on end clients).
- Replace key suppliers.
- Add languages and other localization measures to products and services.
- Experiment for certain groups with new features, services, pricing models, and so on.
- Capture and/or spin "giant" opportunities (e.g., Amazon's entry into cloud computing or Slack's turning an internal tool into its product).
- Leverage human resources flexibly anytime, anywhere (e.g., hire people part time, have a work-from-home policy, allow new fathers to work just three days a week for two years).

Keeping such a visions list — updating it and reminding ourselves of it — is a good cultural measure that the agilification leaders should manage.

> *The main value of being increasingly agile is to allow the organization to realize its potential visions more quickly, with less investment, and with greater chances of success.*

Step 3: Build Abilities That Allow for Visions

The main value of being increasingly agile is to allow the organization to realize its potential visions more quickly, with less investment, and with greater chances of success. To realize those visions, one must have abilities — both non-digital and digital (see Figure 2). Non-digital abilities (which are not the focus of this article) include generic abilities like brand, financial resources and stability, long-term planning, market share, and industry-specific abilities like core technologies, expertise, processes, business models, partner networks, and so on.

Digital abilities are information systems–enabling abilities that allow for agility. Let's define digital abilities through a partial list of key abilities in the form of technologies, attitudes, and approaches we should adopt to become agile:

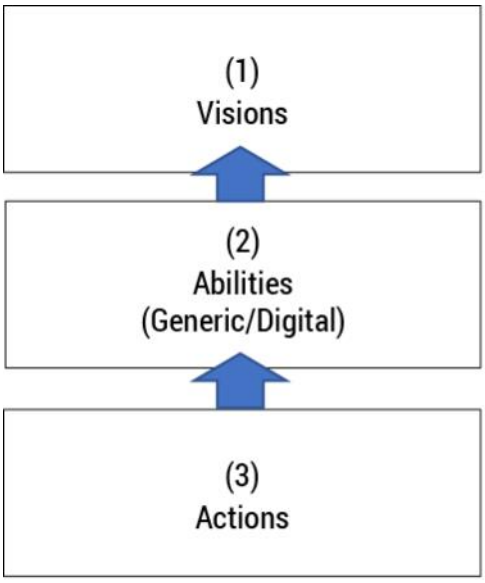


Figure 2 — The relationships among visions, abilities, and actions.

- **Cloud-ness** — key to both experimentation and scaling. Beyond the many advantages of cloud computing, cloud-ness has direct value for agility, including the ability to experiment, to share your applications globally, and to scale.
- **Online-ness** — moving to more online and event-driven approaches and less "batch" work (e.g., a data warehouse, or a system that stores data for analysis, can be replaced with all data being online, all the time).
- **Data-ness** — capturing, storing, harmonizing, analyzing, and gaining business value from data. Data is the new oil of the modern organization.
- **DevOps-ness** — merging development with operations to allow faster changes.
- **User-ness** — knowing your employees, virtual employees, suppliers, customers, future customers, and others by name and ID to allow data to be connected to them and to help data analysis resolve their challenges.
- **Experimentation-ness** — allowing for A/B testing of new features.
- **Automatic testing–ness** — seeking the ability to test your systems quickly to allow for high quality with rapid changes and deployments.
- **Responsive-ness** — developing once for multiple interfaces (i.e., Web, phone, watch, TV, voice).
- **Measure-ness** — measuring how people use your systems and remembering to balance measurement with analysis and action (we often see a lot of measurement but very little analysis, with the business taking very little action as a result of those measurements).
- **Stack-ness** — decoupling layers in your systems (e.g., base systems, data buses, and client systems).
- **Lifecycle-ness** — designing the lifecycles of new systems and planning for the end of life of new systems (including obsolescence).
- **Open-ness** — looking for external technologies and tools that can help accomplish tasks. Work that used to cost thousands of dollars to accomplish can today be done almost for free with advanced tools, plug-ins, open software, cloud, and so forth. The use of open source, microservices, containers, APIs/Web services,

and an event-driven architecture allows faster development, greater flexibility, and speedier and reusable deployment.

Step 4: Initiate Actions That Build Abilities That Allow for Visions

Digital abilities such as those presented in Step 3 (and Figure 2) should be groomed using specific actions. Such actions — in the form of projects, initiatives, and directives — build systems, processes, and skills as well as provide visibility into what you can expect from the ability.

Actions are specific endeavors that both create business value and build long-term abilities. Let's consider a few examples:

- **Harmonize analytics of your websites.** Ensure all your websites have analytics (e.g., Google-based), and you have a process to gain business value from analytics. This is a great start to building data ability and deriving insights into your customer experiences.
- **Move systems X,Y,Z into the cloud.** Select a few internal systems and a few external client systems to move to the cloud. This can start with a survey of systems and looking for solutions that are software-as-a-service (SaaS)-based. This is a good venture into understanding the pros and cons of cloud computing.
- **Train a few teams in Agile methods** (e.g., Scrum, continuous integration, continuous deployment). This is a helpful path toward building the "people" side of agile.
- **Initiate one or two projects that connect business and IT,** both of which work on the project(s) in the same physical/virtual space and experiment with rapid prototyping and development sprints.
- **Train a few teams in design thinking** to create and introduce innovative products and services.
- **Review your systems (and their owners).** This is a great start to mapping the current architecture and designing an updated one.
- **Undertake globalization or localization of some systems.** This is a good way to create a standard infrastructure.

Step 5: Master the Interplay Among Leadership, Culture, Business Architecture, and Digital Architecture

We have found that building agility calls for mastering the interplay among four forces (see Figure 3):

1. **Leadership.** Agility must be driven from the top, mainly because it will call for ongoing culture adaptation and conflict resolution (mainly in the form of who is in charge of what). The leader must fully understand the agilification journey. The leader's decisions must demonstrate the right balance between the organization's current state and the future to-be state, and between small and large, slow and fast, and value and risk. As Step 1 suggests, the forces against agility will be strong, and only clear and assertive leadership can build the needed momentum.
2. **Culture.** Culture is the way in which the organization acts, the principles that ultimately lie behind the ability of individuals and teams to create value. In a sense, agility is a facet of culture — as we want every part, every individual, every effort to be agile. It is up to the leader and his or her lieutenants to set the culture — often called "enculturation" — by explaining, demonstrating, giving direct feedback, encouraging experimentation, promoting minimal viable products (MVPs), and appreciating and building enculturation opportunities.

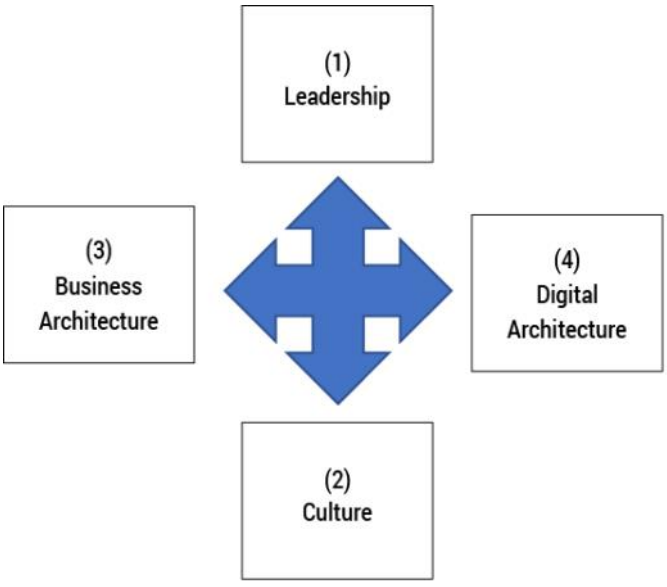


Figure 3 – Master the interplay among leadership, culture, business architecture, and digital architecture.

3. **Business architecture.** This force defines the current, near-term, and long-term futures of the business. It captures the short-term opportunities; plans for next products, business models, or markets; and ponders further long-term value. Business architecture must coordinate efforts to prioritize, plan, and push for implementation of the needed visions (see Step 2).

4. **Digital architecture.** The counterpart to the business architecture force is the digital architecture force, which is entrusted with setting the appropriate evolving technical architecture of the firm. This evolution of the classic discipline of IT architecture manages relations among all current systems, their level of maturity and their place in the lifecycle, and their future.

The key term is *interplay*. To enhance agility, you must understand the nature of these forces. Harmonized interplay will push the organization forward, while harmful interplay will drag the organization down and turn the agility efforts into a battleground. As the leader of the agilification journey, you need to carefully balance these four forces in accordance with your organization's level of maturity and style.

Step 6: Shift to Deadline-Driven Smaller Projects

If there is one measure that can drive an organization toward agility, it is the shift to deadline-driven smaller projects (see Figure 4). Simply put, we recommend defining smaller projects (that can relate to each other to build a bigger project) and focusing on the deadline — not just on the results. This is part of the skill of project management with a drive to flexibility. The key idea: it is better to have 90% ready this month, on time, than 100% ready next month, which is too late. Naturally, one must select the MVP and adhere to the critical path (e.g., FDA regulations, critical features, or connectivity).

Many organizations are already using quarterly and yearly planning cycles. Being focused on deadlines has known advantages. Deadlines can be yearly, quarterly, monthly, weekly, or even daily. Setting

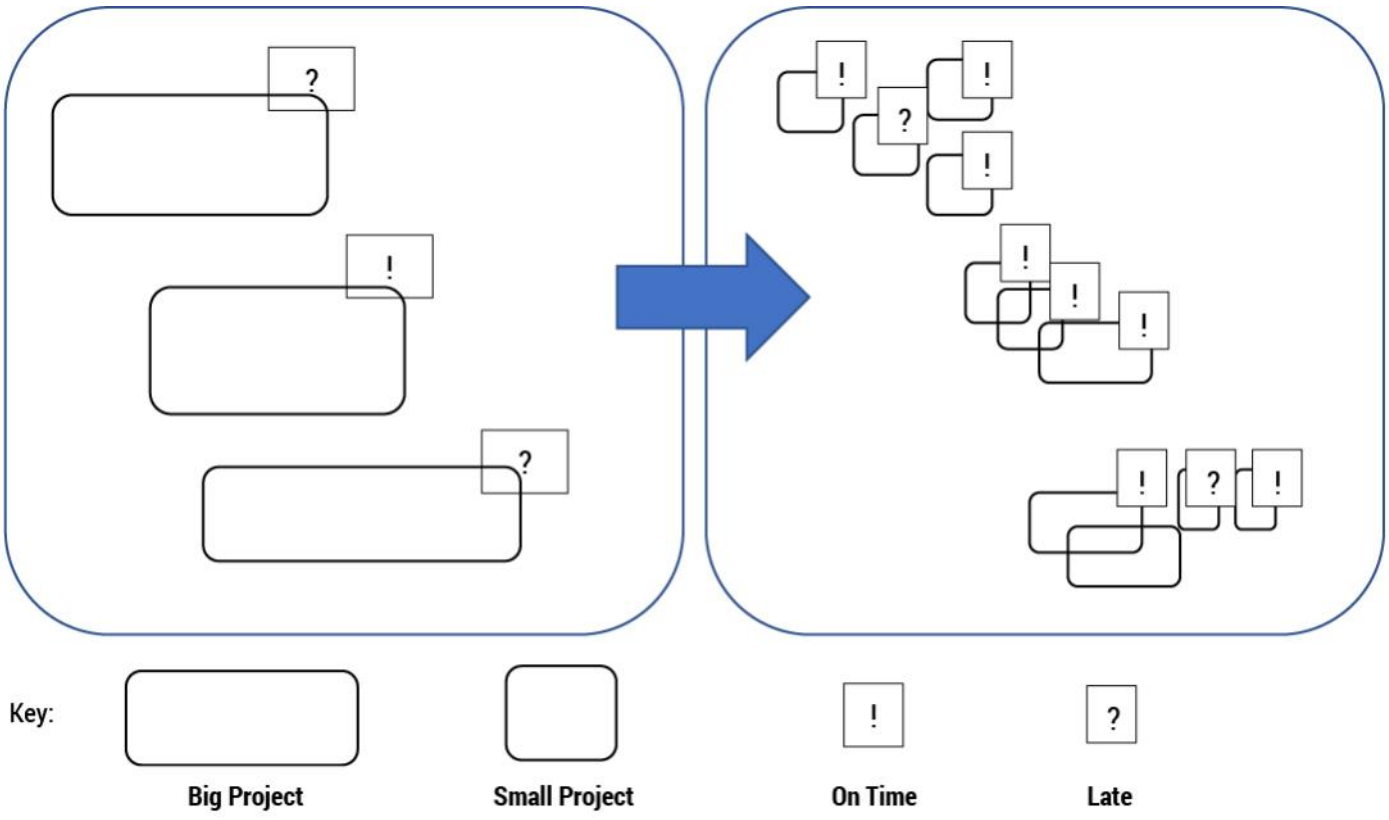


Figure 4 — From big projects that are often late to smaller projects that are mostly on time.

deadlines encourages direct planning, risk analysis, and resource allocation. In this approach, deadline-driven means that:

- Large projects are broken down into smaller parts. Short-term parts should be defined; longer-term items can be defined later.
- You focus on deadlines, not just results ("prefer partial results on time").
- You make sure you have the needed resources.
- You define your goals in concrete terms.
- The task of planning, which takes resources, has a deadline.
- Testing, feedback time, migrations, and so on, can all have deadlines.
- You promise less and try to do more.
- You accept the possibility of missing the deadline, and when you know you are not progressing to schedule, announce the failed deadline early.
- You set review, test, and share meetings in advance (e.g., for the entire quarter).
- You manage risks.
- You use a "cyber-by-design" approach (in which security, privacy, and similar considerations are part of the initial design) to be cyber compliant from the beginning and not just at the end of the development process.
- You start with the "difficult" things. For example, if you are to develop an algorithm and a user interface, begin with the algorithm (assuming there is a risk that you may not have the right solution). Choose to do first the tasks at which you may fail.

The value of deadlines, and especially of smaller chunks of work (as seen in the "release often" philosophy of MVPs), is shown in:

- Observable results
- Ability to adjust to obstacles and changes much more quickly
- Clients/customers being able to see the value being produced on an incremental basis, which allows them to adjust their desires/expectations (if they like one aspect, they can state it; if they feel that something is really wrong, they can say so)
- Building the system for change (e.g., reuse of code, component replacement, auto testing)

Deadline-driven does not mean the abandonment of ambitious goals. When we are deadline-driven, we are simply choosing to have 75% of the desired results delivered on time, rather than 99% delivered late (there is usually a small percentage that is not delivered). Seventy-five percent of the results is usually 90% of what is needed, and if the other 25% is really needed, we set a new deadline.

In the long run, for smaller chunks of work, *learning is the best benefit of being deadline-driven*. Such shorter cycles allow for more wins, as well as more fails — encouraging (if there's enough time allocated) learning, which is critical for building and evolving an agile culture.

The pressure to be agile stems both from the fear of being disrupted and the great opportunities that the digital world enables.

Conclusion

Being agile is among the key core qualities of the modern organization. If your organization is not yet agile and you are in an industry adjacent to an already agile industry, you have an even greater challenge: how to become agile fast and not become disrupted. Frankly, in some cases, this may be impossible (think Kodak in the age of digital cameras).

The pressure to be agile stems both from the fear of being disrupted and the great opportunities that the digital world enables. Bear in mind that the pace of M&A is likely to rise due to (1) the digital investments that call for larger organizations and, conversely, (2) the ability of tiny players to act big (and thus disrupt established players) because of cloud computing.

The six steps presented in this article are designed to help guide agilification leaders, especially as they take their initial steps in the agilification process:

1. Appreciate the **mental challenge** of agility as "building flexible buildings."
2. Develop and **share visions** of what to expect from agility.
3. **Build abilities** that allow for visions.
4. **Initiate actions** that build abilities that allow for visions.
5. Master the **interplay among leadership, culture, business architecture, and digital architecture**.
6. Shift to **deadline-driven smaller projects**.

The imperative of the agile organization is directly linked to the digital force. On the one hand, we *must* be agile because of the external market and customer expectations of digital transformation; on the other hand, we *can* be agile because of internal digital technologies. Finally, despite difficulties, we need to remember that agilification has its benefits, too: it makes work very interesting — and very different. Enjoy the journey.

Yesha Sivan is the founder and CEO of i8 ventures, a business platform focusing on "innovating innovating." He is also a visiting professor of digital, innovation, and venture at the Chinese University of Hong Kong Business School. Dr. Sivan's professional experience includes developing and deploying innovative solutions for corporate, hi-tech, government, and defense environments. He focuses on digital strategy (SVIT – Strategic Value of Innovation Technology), innovation and venture (employment black holes), mindful leadership (orange bike workshop), virtual worlds (3D3C platforms), and knowledge age standards (nine keys). Dr. Sivan earned a doctorate from Harvard University and has taught EMBA, MBA, engineering, and design courses in his areas of expertise. He can be reached at yesha@i8.ventures or via his blog (http://www.DrYesha.com).

Raz Heiferman is a Senior Digital Transformation Advisor at i8 ventures. Previously, he was Co-CEO and Senior Advisor at Be Digital; Acting Government CIO and Manager of the Shared Services Division at the Israel CIO Office; CIO at Direct Insurance, Bezeq, and Optrotech; and has held other senior positions in two leading Israeli software houses. Mr. Heiferman teaches graduate courses on business strategy and digital transformation at several academic institutions and has published five textbooks on data warehouses, SQL databases, and file organization, along with many articles for Israeli professional magazines. He is a Certified System Analyst and has served as Presidium Member of Israeli Information Technology Chamber. Mr. Heiferman holds a bachelor's degree in economics and statistics and an MBA (cum laude with operations research specialization), both from Hebrew University, Jerusalem. He can be reached at raz.heiferman@i8.ventures.

THERE'S GOTTA BE A BETTER WAY

No More Snake Oil: Architecting Agility in a Complex Environment

by Barry O'Reilly and Gar Mac Críosta

The confusion surrounding the role of architecture when aiming for agility isn't simply a labored talking point — it's part of the reason Agile initiatives fail and architecture teams are losing influence. As it stands, it appears Agile and architecture are struggling to find a fit. This article considers the possible effects of a third way: agility through "antifragility." Rather than aiming to control, or to remove control, we should seek to build systems, both technical and business, that aim to be antifragile to change. This allows the production of business and technical architectures that enable agility through design rather than process or mindset. Taking ideas from systems engineering, complexity science, and recent survey data, we explore how the inherent interconnectedness of architecture and agility can be leveraged — via the Antifragile Systems Design process — to make the management of complexity something all organizations can do.

Enterprise Software and VUCA: The Need for a New Approach

The modern business environment is a strange place, if visited by the manuals and best practices of yesteryear. The end of Taylorist management science[1] is, according to some, clearly in view.[2] Indeed, the complexities of the modern world refute the join-the-dots MBA business playbook. The world of VUCA (volatility, uncertainty, complexity, ambiguity)[3] requires a new approach. Disintermediation, globalization, market upheaval, disruption, and technological advance all combine to produce an effect that is difficult to mitigate, impossible to predict, and arduous to detect. The software crisis,[4] first defined in 1968, is entering a new phase, and the consequences of continued shoulder shrugging are becoming ever more serious.

Witness the growth of the Agile industry, with its ceremonies, high priests, and rituals. It has, quite rightly, found the zeitgeist: the decline of management science and the pseudo-scientific pretense of order in the domain of complex human systems. This is what causes Agile mysticism; we know that waterfall will not work, so we reject it based on past experience but do not replace it with anything demonstrably better. This creates the gap for "snake oil." The diagnosis of the multiple failings of waterfall is completely correct; yet the results of the Agile cure do not seem to bear the weight of investigation.

A 2017 report of 300 UK-based CIOs demonstrates the problem: 21% of Agile projects end in complete failure (i.e., nothing delivered), and 68% of CIOs want to see more architects involved in Agile projects.[5] Moreover, the projected cost of Agile failure is 37 billion British pounds (US $48 billion). Yet, a recent IASA Global survey[6] reveals that over 75% of 260 responding organizations are implementing some form of Agile practice, and 50% are implementing Agile-at-scale. However, less than 50% of all respondents have integrated architecture into their Agile process.

In an environment where both inflexible and unstable software can lead to business failure, modern businesses need both the flexibility espoused by Agile practitioners and the rigor of more structured systems engineering methodologies. This contention is the source of much debate and confusion between the Agile and architecture camps and requires an alternative architectural approach. Thus, we propose that by *architecting for antifragility*, businesses can gain real agility and deliver systems with a higher level of quality. NYU Distinguished Professor Nassim Nicholas Taleb describes an antifragile system as one that gains from disorder; a system that becomes stronger when exposed to stressors (even unpredictable or unknown stressors).[7] An antifragile system is by definition agile and resilient.

Accepting Complexity

Complex systems, under which most contemporary business-critical systems would be classified, are not merely complicated. They are systems that cannot be

assumed to behave in a certain way and have nonlinear responses to changes in input. Consider the concept of the Platonic fold,[8] which tells us that the act of modeling the world simplifies it to the point where any decisions made based on that model are misinformed due to details omitted for the sake of hiding complexity. Thus, dynamic real-world problems twist and bend, while the static solution cannot keep up, causing the demise of quality. In software, this leads to a multitude of problems, including shortened life span, patching, and quality issues.

When humans build complex systems, they tend to fail, often catastrophically, because of Platonic folding. The solution to the Platonic fold requires accepting complexity as something we can neither predict nor control, along with accepting the limitations of modeling and risk management. Instead of pursuing correctness in these areas, we should aim to build systems that are antifragile to fluctuations in the VUCA elements (i.e., the system becomes stronger as the business environment warps and changes with time).

Dynamic real-world problems twist and bend, while the static solution cannot keep up, causing the demise of quality.

Antifragility in Software

Due to extensive research being carried out on the subject of computational antifragility, many solutions to this kind of problem will emerge in the future.[9] It is important to realize that the degree of fragility of a system is often a function of its internal structure. The ability of a system to change under stress is governed by the interconnectedness of its parts, how strongly they are tied to each other, and how much change ripples through the system. Therefore, there is a need to ensure that we match the level of interconnectedness of a system's components with the effort required to reorganize them in the face of change. This is something that architects are well qualified to do.

For many years, the decomposition of software systems has been held captive by the latest technological trends, vendor interests, and a slow-shifting mindscape. Many students of software engineering still hold fast to ideas of elegance and reuse, often making software unnecessarily complex in the process. There has, however, been a broad library of dissent against these methods, dating back to 1972. Software engineering pioneer David Parnas's ideas on nonconventional decomposition[10] tell us that we can build better systems by focusing on what will change rather than what will happen functionally, while software architect Juval Löwy's important distinction between functional- and volatility-based decomposition via the IDesign Method[11] provides some ideas and techniques that make this easier.

Each of these methods relies on focusing on the elements that can change, rather than on concrete requirements. By building a system where the primary requirement is the ability to handle change, a very different piece of software is constructed than would happen otherwise. This need for change in design philosophy — away from building to specific requirements and toward building systems that are antifragile — has been expressed elsewhere, including at NASA.[12] Kjell Jørgen Hole's book *Anti-Fragile ICT Systems* illustrates that systems demonstrating high levels of antifragility have the following four properties:[13]

1. Modularity (consisting of separate, linked components)
2. Weak links (a low level of interconnectedness between components)
3. Redundancy (the presence of more than one component to cope with failure)
4. Diversity (the ability to solve a problem in more than one way with different components)

Antifragile Systems Design

The Antifragile Systems Design process guides the architect to optimize and balance the four antifragile properties mentioned above with the VUCA elements present in a project. With a few days of analysis and design work, we can shift any project in the direction of antifragility, without incurring a great deal of overhead. The Antifragile Systems Design process mixes ideas

from complexity science and systems engineering to create a method to guide the design effort.

This process embraces the complexity in building dynamic systems. Following the advice of Taleb, Parnas, Löwy, and others, we need to focus on what we do not know before focusing on what we do know — accepting our limitations and our inability to predict the future. Indeed, the Antifragile Systems Design process is not fixed but can grow and change with every project. With this new architectural approach, the intention is not to create yet another framework or silver bullet, but to provide a starting point for a new type of design process. This process follows several simple steps and requires no more tooling than an Excel spreadsheet.

Who Takes This On?

The steps outlined below require a mix of skills within business, business architecture, and software engineering. However, this is not simply a business activity or a software design activity and cannot be divided into different tasks for different silos; each step in the process creates feedback loops to ensure that answers arrived at are coherent. Antifragile Systems Design requires an organization to move as one toward solving the problem of complexity, which means changing the perspective from "us versus them" (IT versus business) to simply "us" (business). Business leaders, business/enterprise architects, and software architects all need to engage with the process to make it work. This requires a new approach from both architects and business leaders.

Architects need to work with the business to describe the VUCA environment, translate the impacts on the software decomposition, and even assist in business-level mitigations. Currently, few architects span this range; therefore, a business architect and a software architect often must work together to guide the process. However, it is possible for a single architect (business/architecture-focused or software-focused) who combines business understanding and software engineering knowledge to guide the process.

Business leadership plays an important role in enabling the architects and the project to embrace this approach. By employing Antifragile Systems Design at a high level, business leaders can learn to ask the right questions of their software teams and quickly assess the stability of an initiative.

Step 1: VUCA Analysis

In the first step, we describe the VUCA environment for this particular initiative, listing the VUCA elements with regards to the business model, and begin to sketch our architecture. We design the system to cope with fluctuations based on the VUCA elements identified in the business model, meeting each challenge with a change in one or more of the four antifragile properties of the system.

This exercise starts at the business level, with input from business leaders. It identifies VUCA elements in the business model and clarifies what business mitigations, if any, are in place or need to be in place. This step can actually help improve the business processes or organizational structure behind the initiative. This kind of work is usually carried out by the business, but rarely shared in detail with architects. VUCA analysis requires the following actions:

- Represent the initiative's business model using the Business Model Canvas[14] and its standard building blocks.
- Perform a VUCA analysis, noting everything considered volatile. For example, what can change? What happens if a partner is acquired or ceases trading? What happens if a cost escalates? This is a useful exercise for the organization and can educate the architect in how the wider market works.
- Run through everything that is uncertain. For example, what do we not know? What is purely guesswork? What impact can a lack of knowledge have on the system?
- Run through all complexities (processes that have nonlinear responses to input) and ambiguities. Explore the impact of being wrong about something and what would need to change to accommodate the error.
- Record this in a spreadsheet, with a list of VUCA elements and the corresponding mitigations.
- Choose the most appropriate mitigation for each VUCA element, excluding those too expensive or unrealistic.

- Continue the exercise until the mitigations start to become repetitive.

Note that this exercise does not involve trying to predict the future, but rather having an awareness of the types of change that can happen to a system. We cannot predict all change, but we can work with what we know.

Step 2: System Decomposition — Flow First Design

The next step is to propose a system design. Here, we use Black Tulip's Flow First Design, a design process for distributed systems, described briefly below:

- Describe the software as a series of data flows enabling the functional requirements.
- Create a component decomposition for each flow that is completely decoupled from all others and all data sources; the flow is its own system. This creates a system with very low levels of interconnectedness.
- Subject each data flow to the fluctuations described in the VUCA analysis.
- Ensure that the mitigations listed in the VUCA analysis are represented in the software.
- Consolidate different flows, reducing the level of interconnectedness; aim for minimum disruption when each VUCA element changes, as described by Parnas.[15]

This allows the architect to refine system decomposition by measuring the system's ability to meet changes likely to happen based on the VUCA analysis. The system decomposition now relates to both functionality and system behavior. This step establishes the right level of modularity and weak links, the first two properties of systems demonstrating high levels of antifragility, and connects them to the VUCA elements identified previously.

This step requires knowledge of software engineering patterns and the management of coupling; however, it does not require a detailed knowledge of software development. It is enough to be able to ascertain that a VUCA fluctuation will have a minimal level of impact on the system.

Step 3: Design Testing

In this step, we present the architecture to various stakeholder groups through an exercise such as the Architecture Trade-Off Analysis Method (ATAM).[16] This ensures that all concerns have been addressed and that the VUCA analysis was accurate and promotes confidence in the role the architect has played by providing a sense of rigor and demonstrating a potentially robust and resilient system.

Step 4: Modified FMEA

Failure Mode Effects Analysis (FMEA)[17] is a Six Sigma technique that helps manage quality in a system by investigating how the system will cope with failure. Using FMEA, we can investigate system behavior and adjust the architecture to be resilient to failure during operations. However, in this step, we do not attempt to prioritize or predict risks or criticality, as this provides little benefit when dealing with complex systems. FMEA includes the following actions:

- Create a FMEA spreadsheet listing the different ways each component can fail.
- Record how failure is detected and mitigated and the impact of component failure.
- Aim for a high level of automation.
- Change the system design to accommodate mitigation of these failures.
- Repeat the process for any number of failure modes until the mitigations become repetitive.

This step in the process tunes the system to have the right balance of redundancy and diversity (the last two properties of systems demonstrating high levels of antifragility), pushing the system toward antifragility. This step also protects against the risk that too many mitigations can produce an overcomplicated system. In such a case, FMEA will struggle to mitigate all known errors at a reasonable cost and will send the architect back to the VUCA analysis for a more realistic take on what can change or to the decomposition step to redraw the system scope.

Why This Process Works

To make this process work, we can leverage the idea of *exaptation*,[18] where an element of a system developed for one purpose can have serendipitous effects for another purpose. Building a wall in your house, for example, allows spreading the load of the roof, but also provides the basis for rooms, stops noise traveling between rooms, and gives privacy. A wall also stops fire from

spreading, provides somewhere to hang paintings, and a place to bang your head against when dealing with Agile coaches. When we combine two separate mitigations, say the wall and the fact that we added a space in the wall for insulation, we suddenly create the conditions for dealing with something we did not see coming — hiding electrical wires in the wall!

In working through the list of VUCA elements, tweaking the design, and adding mitigations, with each mitigation the system becomes antifragile to that particular VUCA element. The first 10 are usually tricky, but after 50 mitigations, a pattern emerges: many of the VUCA elements in the list are resolved by previous mitigations and the effect of mitigations can be said to be nonlinear. By following this process, the system trends toward antifragility, which is the only possible good result in a complex environment that we do not control. When this process repeats as part of the FMEA step, the likelihood of future exaptation increases. The VUCA analysis also builds confidence among stakeholders that the system will be "robust," but, as architects, we know that we are doing much more than that: we are providing the bedrock for antifragility! We call this pattern *nonlinear system responsiveness*.

Once a system is in place, the Antifragile Systems Design process becomes iterative. Every failure is considered feedback and the system should be strengthened by the team by rerunning the process. The best example of this kind of system is Microsoft's Azure or Amazon's AWS cloud platforms — outages are used to strengthen the platform, with these two platforms becoming some of the most resilient in the world.

While the idea of nonlinear system responsiveness seems intuitive, it has as of yet no proven mathematical basis and is not guaranteed to occur. However, by aiming to induce it, we at least make the system less fragile and provide the basis for a positive, nonlinear response. The actual degree of exaptation can never be predicted and will never be complete (all systems will die someday), but this process actively encourages exaptation as the premier focus of the design effort.

Concrete Actions for Business Leaders

Going forward, business leaders should consider the following actions:

- Understand that complexity is the key cause of software failure.
- Don't waste time and take unnecessary risks by trying to predict and control the unpredictable and uncontrollable.
- See software execution as a business task with varying results that requires constant monitoring beyond status reports.
- Use VUCA analysis to understand the stability of IT delivery. "What happens if?" questions tell you all you need to know about a software project's quality. Bring the architect into the core business team and make VUCA analysis a natural part of your execution.
- Enable your architects to embrace antifragility as the key to real agility.
- Understand that current industry trends around Agile cannot deliver in the face of complexity. Use the VUCA analysis process to have a voice and influence in the direction of software projects and ensure quality is there from the start.
- Demand traceability in architectural decision making.
- Ensure that technical decisions are grounded in a shared understanding of the VUCA environment and are FMEA-tested.

Concrete Actions for Architects

Going forward, architects should consider the following actions:

- Practice VUCA analysis on the initiative's business model. A thorough grounding in business basics is required, which can be a challenge for technically focused solution architects. This is a necessary evolution of the role of the architect and cannot be avoided.
- Become an expert in software decomposition.
- Learn different methods for software decomposition, the difference between service-oriented architecture and microservices, the IDesign Method, and Flow First Design. Learn how modern cloud applications are composed and the major components involved.
- Learn to use modified FMEA to improve system designs.

Conclusion

The result of this work is a business with a better understanding of its own fragility and a software system capable of bending and meeting the needs of the changing business environment. This kind of process calls for a new type of architect and a new type of architecture. It requires a solid understanding of the business environment, the effects of change on the business architecture, and a thorough understanding of how software can be decomposed, rather than written. This cross-set of skills can allow architecture to contribute by designing antifragile systems that enable agility and answers the business question of how to become resilient to the VUCA world.

There is no guaranteed result from this process, so the Taylorist approach of measurement, prediction, and comparison will not provide any benefit here. Over time, this approach will succeed for some and fail for others, and this lack of certainty may cause many to resist the approach. The alternative — to do nothing and wait for machine learning and complexity science to solve problems — is not a viable option for today's enterprises.

Acknowledgments

Many thanks to Dr. Riccardo Bennett-Lovsey and Tanya O'Reilly for their valuable comments and suggestions on the drafts of this article.

Endnotes

[1]"Taylorism." *Encyclopaedia Britannica* (https://www.britannica.com/science/Taylorism).

[2]Stacey, Ralph D. *Complexity and Organizational Reality: Uncertainty and the Need to Rethink Management After the Collapse of Investment Capitalism*. 2nd edition. Routledge, 2010.

[3]Bennett, Nathan, and G. James Lemoine. "What VUCA Really Means for You." *Harvard Business Review*, January-February, 2014 (https://hbr.org/2014/01/what-vuca-really-means-for-you).

[4]"Software crisis." Wikipedia (https://en.wikipedia.org/wiki/Software_crisis).

[5]Porter, Chris. "An Agile Agenda: How CIOs Can Navigate The Post-Agile Era." 6point6, April 2017 (https://cdn2.hubspot.net/hubfs/2915542/White%20Papers/6point6-AnAgileAgenda-DXWP.2017.pdf).

[6]Mac Críosta, Gar. "IASA State of Architect Engagement 2018." LinkedIn, 20 August 2018 (https://www.slideshare.net/Garmaccriosta/iasa-state-of-architect-engagement-2018-prelim).

[7]Taleb, Nassim Nicholas. *Antifragile: How to Live in a World We Don't Understand*. Allen Lane, 2012.

[8]Taleb, Nassim Nicholas. *The Black Swan: The Impact of the Highly Improbable*. 2nd edition. Random House, 2010.

[9]De Florio, Vincenzo. "Antifragility = Elasticity + Resilience + Machine Learning Models and Algorithms for Open System Fidelity." *Procedia Computer Science*, Vol. 32, 2014 (https://www.sciencedirect.com/science/article/pii/S1877050914006991).

[10]Parnas, David L. "On the Criteria to be Used in Decomposing Systems into Modules." *Communications of the ACM*, Vol. 15, No. 12, 1972 (https://dl.acm.org/citation.cfm?id=361623).

[11]Löwy, Juval. "Volatility-Based Decomposition." IDesignIncTV, 22 November 2013 (https://www.youtube.com/watch?v=VIC7QW62-Tw).

[12]Jones, Kennie H. "Engineering Antifragile Systems: A Change in Design Philosophy." *Procedia Computer Science*, Vol. 32, 2014 (https://www.sciencedirect.com/science/article/pii/S1877050914007042).

[13]Hole, Kjell Jørgen. *Anti-Fragile ICT Systems.* Springer, 2016.

[14]"The Business Model Canvas." Strategyzer, 2018 (https://strategyzer.com/canvas/business-model-canvas).

[15]Parnas (see 10).

[16]Kazman, Rick, Mark H. Klein, and Paul C. Clements. "ATAM: Method for Architecture Evaluation." Technical Report, Software Engineering Institute/Carnegie Mellon University, August 2002 (https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=5177).

[17]"FMEA — Failure Mode and Effect Analysis." Six-Sigma.se, 2007 (http://www.six-sigma.se/FMEA.html).

[18]"Exaptations." Understanding Evolution, 2018 (https://evolution.berkeley.edu/evolibrary/article/exaptations_01).

Barry O'Reilly is the founder of Black Tulip Technology and creator of Antifragile System Design. Previously, he held positions as Chief Architect for Microsoft's Western Europe practice and IDesign, IOT TAP Lead for Microsoft's Western Europe practice, Worldwide Lead for Microsoft's Solution Architecture Community, and startup CTO. He can be reached at barry@blacktulip.se.

Gar Mac Críosta is the founder of Business Model Adventures and leads IASA Global's Next Architecture Practice Group. He has facilitated the development of change programs with C-level executives, senior managers, technology leaders, and executives in the areas of business model innovation, digital strategy, architecture, and organizational effectiveness (Lean/Agile) across various industries. Mr. Mac Críosta is a Certified Architect Professional (IASA CITAP), a Fellow of the Irish Computer Society, and a LEGO Serious Play Practitioner (LSP). He can be reached at gar@businessmodeladventures.com.

CONQUERING THE HILLS

Agile Architecture or Architectural Agility?

2 Fundamentally Different Paradigms Come Together

by Jan-Willem Sieben, Jan-Paul Fillié, and Cristina Popescu

Nowadays, there is a huge popular demand for Agile as a means to enable change and accelerate value. Popularity, however, is something other than reality; for most companies, the introduction of Agile requires a significant mindset shift. This almost always meets resistance from several directions in the organization. In addition, Agile adoption is often accompanied by some element of inefficiency and chaos if left unguided.

In contrast, enterprise architecture (EA) suffers from a decreasing reputation in technology innovation. This reputation can be the result of certain poor practices: in most organizations we encounter, architecture mainly focuses on its function as a design authority or regulating body. In this sense, it is often seen as an inhibitor of change instead of an accelerator — the infamous "architecture police."

This article describes in more detail these and other common pitfalls and bad practices and tries to identify the pros and cons of both Agile and EA to find ways that the strengths of one can help prevent the weaknesses of the other. We begin by highlighting some bad practices or common mistakes when introducing and operating Agile and EA. These bad practices tend to appear and reappear and are difficult to root out; we therefore use the term "anti-pattern," as introduced by Andrew Koening,[1] to label the bad practices. Wikipedia defines an anti-pattern as "a common response to a recurring problem that is usually ineffective and risks being highly counterproductive."[2] Next, using real-life examples from client work, we show the advantages available to organizations that successfully combine Agile and EA.

Challenges in Enterprise Architecture

The past 30 years have seen the rise and maturation of enterprise architecture. Beginning in 1987 with John A. Zachman's Information Systems Architecture model,[3] the EA discipline has since seen many advocates, research, and frameworks worldwide. Its foremost promised value: improving the efficiency of IT assets and the return on IT investments by designing, improving, and managing the complex landscapes of information systems.

Nevertheless, over the last 10 years, several published studies and opinions indicate that the value of EA may be overrated, and that EA might not deliver on its promise or might even be considered "dead," as Australian columnist Jon McLeod has stated.[4] Further extensive research, such as from the Erasmus University Rotterdam, also leads to a somber conclusion that two-thirds of EA projects fail.[5] Even though this is probably "at par" with all other IT-related endeavors, it is an aspect we need to improve.

As we take a closer look at the reasons why EA has fallen short of expectations, it is valuable to highlight two anti-patterns that we have encountered repeatedly with clients.

EA Anti-Pattern 1: Process over Value

The first EA anti-pattern is apparent when enterprise architects are more concerned with correct adherence to the architectural process, often defined by their own derivative of an EA framework (e.g., TOGAF). Of course, this process was designed to ensure the maximum effectiveness and value of architectural work, but the focus should always be on the value part of the equation. The tendency to simply follow the process, designed in the past and often inherited by generations of enterprise architects, is too strong.

In practice, the enterprise architects are then more concerned with the correct "paper trail" of architectural documents, design and solution architectures, and, the most dangerous TOGAF template of all, the "Architecture Requirements Specification."[6] This limits the effectiveness and responsiveness of the EA team and, most importantly, erodes the perceived business value of EA in general.

EA Anti-Pattern 2: The Disconnect of the Ivory Tower Architects

The second anti-pattern occurs often in large organizations where the EA function has been recognized as important but resides as a staff organization — typically part of the office of the CIO. Because of its "staff" label, the enterprise architects' impact and mandate are limited; managers can avoid or ignore architects to a great extent. Especially when the informal culture is strong and decision making is, in effect, local and lies with project managers, the enterprise architects become increasingly detached from the day-to-day practice of projects, programs, investment boards, and even strategic discussions on the mid- and long-term future of IT. They do not contribute visibly to successes and failures and hence cannot be held accountable. When this endures, the enterprise architects will be practically disconnected from the (informal) IT powerhouse, will be unable to deliver solid and trusted advice, and will have no mandate over strategic IT discussions (see sidebar "The Ruins of the Ivory Tower").

Challenges in Introducing Agile Practices

Agile is a mindset. The overall concept of Agile is about continuously getting better at whatever it is we are doing. The practices and ways of working related to an Agile mindset have been well proven as more effective and efficient than the traditional waterfall methods — across IT and other business functions alike. Not only do they help create more value, but they also cultivate more candid and authentic workplaces. Looking for benefits such as faster time to market, better-quality products, and more engaged employees, organizations of every size are now adopting and maturing an Agile mindset.

However, as companies adopt Agile as their standard for software development, they usually encounter resistance from several directions — from other parts of IT as well as from the business. This resistance is a result of aversion to change, with existing structures and leadership holding on to practices that worked in the past. Indeed, it is quite challenging to expand the first successful pilot projects to an enterprise-scale Agile capability. We often see organizations struggling with cultural change, insufficient business involvement, and other aspects of scaling. In design thinking terms, these challenges are called "hills." The hills model of Agile transformation (see Figure 1[7]) shows the most common challenges that organizations face when applying Agile-at-scale; note: the order in which these hills are encountered is different for each organization:

- **Hill 1: Changing the organizational culture.** Changing the organizational mindset is key for a successful Agile implementation on an enterprise scale.
- **Hill 2: Getting the business involved.** The role of product owners, but also the support of the overall business, is the driving force behind Agile success.
- **Hill 3: Coping with different speeds of change.** Not all parts of the business and not all teams providing support for technology can work and accelerate at the same pace.
- **Hill 4: Extending Agile to the full lifecycle with activities and automation.** Operations can only embrace Agile if nonfunctional requirements are met. This can be done through end-to-end enablement and automation.
- **Hill 5: Scaling Agile.** Collaborating on an enterprise scale requires finding alignment between different teams to cope with dependencies, to share best practices, and to distribute the work effectively.
- **Hill 6: Distributing Agile.** Distribution allows for access to talent and resources across the globe,

The Ruins of the Ivory Tower

A midsized European bank used to develop and maintain all its client-facing applications from a single distribution department. All architecture was created by a single group of architects that was responsible for the enterprise vision and technology direction. The bank's adoption of Agile practices triggered several organizational changes, including a new organizational structure for IT where different product groups were formed to channel Agile development. While the Agile teams in each product group were working from their own backlog and created their own solution architecture, the central enterprise architects were still developing end-to-end architectures for the enterprise on the whole. However, they did not have any mandate or influence on budgets and no product group took notice of their output. This situation continued for two years before the architecture group was disbanded and the architects assigned to different departments.

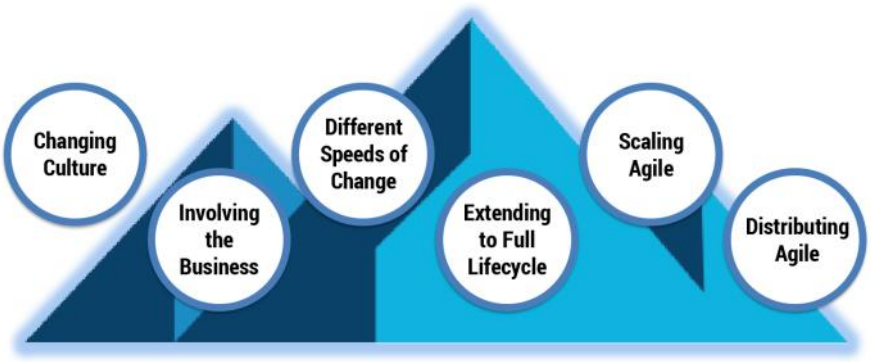


Figure 1 — The hills of Agile transformation. (Source: Fillié and Boer.)

potential cost reduction, and opportunities for improved innovation.

To overcome these hills, some organizations use ways that worked for them in the past, but in an Agile context this results in counterproductive outcomes.[8] As we saw earlier with EA, these anti-patterns are hard to root out and tend to reappear. Below, we highlight two of the most common Agile anti-patterns.

Agile Anti-Pattern 1: Self-Driven Teams Running Wild

For any organization starting with Agile, it makes sense to begin with one or more pilot projects to demonstrate the added value of working in this new way. Practices like retrospectives capture lessons learned and ways to improve. From this first initiative, the organization can then start expanding. Several organizations have found that dispersed initiatives do not necessarily bring added value to the overall business. If each department has its own Agile teams defining their own way of working and making technology decisions without consulting each other and without a common vision, then Agile can soon become a very expensive exercise with only local benefits for the business. After some time, this can result in unwanted internal competition, multiple standards, different ways of working, and an increasingly complex technology landscape.

Agile Anti-Pattern 2: Lack of Enterprise Alignment

This second Agile anti-pattern usually happens in larger organizations where architectural considerations are entirely left to the development teams themselves. Without proper alignment through technical and business architecture, making the best technical and architectural decisions at the product level can result in unintended negative consequences for the larger organizational ecosystem. This becomes especially important in enterprises with large application portfolios, and even more important when a mix of legacy and modern technology needs to be integrated. While the design and execution of each product on its own is very important, ignoring the context of the larger ecosystem where these products will be used is highly risky.[9] Similarly, business processes will be difficult to align and overall cost will increase.

This anti-pattern is a result of pushing decision making down to small self-organizing teams without properly enabling them with the knowledge and support they need to get the job done and a clear business direction to guide them. Architectural spikes and refactoring are not enough because the team does not have sufficient understanding of the dependencies within their organizations.[10] The consequences range from resources spent on overlapping and redundant functionality to significant amounts of rework needed to course-correct when issues surface.

Learning from Each Other: Bridging the Gap

The anti-patterns described above show a clear need for better practical approaches to both EA and Agile. Based on our client experience, a significant opportunity to improve the business value of both lies in combining their practices. Several frameworks could help in this sense, and with the right vision in mind, companies can increase their ROI for both EA and Agile. For example, the use of frameworks like the Scaled Agile Framework (SAFe), that are built around the idea of striking the right balance between Agile and EA, is growing significantly — although with various degrees of

success, to be frank. The balance of "emerging design" and "intentional architecture"[11] is key here, as well as the way the IT organization is structured in terms of roles, responsibilities, and level of cooperation.

In practice, we have seen several ways in which organizations combine EA with Agile thinking and methods to break through the anti-patterns and improve results. The next section highlights four useful examples.

SAFe: Providing the Long-Term View to Agile Teams

The first example is from a Netherlands central government client and comes down to the adoption of Agile architectural approaches described in SAFe 4.5.[12] This organization has adopted SAFe as the standard process and methodology framework for its journey toward more business agility and more efficient use of IT assets. Up until 2017, the organization's IT had increasingly become a liability due to outages, aging software and subsequent support issues, and rising costs. The enterprise architects were dealing with both anti-patterns discussed earlier: they were part of the CIO office, with limited mandate, and were mostly concerned with compliance — both in terms of architecture as well as of the business as a whole — around issues such as security and public safety.

A huge modernization program was launched in 2017 with the general idea of using a greenfield approach to modernize the corporate IT landscape. New applications were going to be deployed, cloud-native, onto a new hybrid cloud: the *innovation domain*. The client's own adoption of SAFe 4.5 was to govern the development and management of the growing portfolio of these applications. Existing applications were analyzed and, if deemed appropriate, migrated to this new hybrid cloud in a separate environment that supported multiple operating systems, middleware, and databases: the *migration domain*. The remaining applications that needed some sort of lifecycle extension due to business requirements were managed using traditional IT methods in the *legacy domain*. The rest (around 15%-20% of applications) would be terminated.

Over a period of five years, this ecosystem was then going to be simplified and, in effect, the legacy and migration domains would be phased out. Table 1 illustrates the domains over time, with their appropriate solution development and governance framework.

The role for architects in these domains (and over time) varies. In the innovation domain, the architects act in three SAFe abstraction layers: on the large solution, program, and team levels. They actively participate in the role of solution architects and solution engineers. They are responsible for the SAFe practices of writing enabler epics and participating in value stream coordination sessions to establish backlogs and architectural runways.[13] They also oversee the nonfunctional requirements, the reuse of solution elements, and the establishment of an architectural repository, where solution elements and patterns are managed. Thus, the architects focus on value instead of process.

However, the top level of SAFe, the portfolio level, is not implemented. At this strategic level, the enterprise architects still must deal with the long-term IT strategy and their current political reality of negotiation, delays, and alliance formation that typifies government policy and decision making. They will keep relying on their current EA approach, largely based on TOGAF. There is a reason for this choice: it is proving quite useful, since during transition (2017–2022), there is a bimodal character of the IT landscape to be managed and the choice has been made to be pragmatic and use the existing way of working to allow the top level of the organization to get used to the cultural aspects of adopting Agile.

Growing their impact in this IT transition period, as illustrated here, will help the enterprise architects add value by identifying cross-team solutions, promoting the reuse of those solutions and patterns (standardization), building the architectural runways, and advising on prioritizing the backlogs on various levels. Moreover, the Agile way of working has facilitated more direct involvement of the enterprise architects with the teams. The architects are more engaged in day-to-day discussions and, because of the increased heartbeat of the IT organization, they have more touchpoints where they can advise and guide IT decisions. Their former stronghold was in effect destroyed for the innovation domain: there was less need for all the blueprints, project start architectures, and other documents and designs up front that they were concerned with before. This meant a shift in thinking and a shift in pace that has proved beneficial so far (the program is still in its turbulent midterm phase).

Timeframe	IT Environment	Project, Portfolio Management, and Governance Management Methods	EA Method/ Framework
Before 2017	Traditional on-premise infrastructure	Mainly waterfall • Prince2* • ITIL/ASL/BiSL*	TOGAF*
2017-2022	*Legacy domain*: on-premise infrastructure	Mainly waterfall • Prince2* • ITIL/ASL/BiSL*	TOGAF*
	Migration domain: hybrid cloud, multiple OS, MW, and DB versions	Mainly waterfall • Prince2* • ITIL/ASL/BiSL*	TOGAF*
	Innovation domain: hybrid cloud, containerized, standardized – cloud native apps	Agile and limited DevOps • SAFe 4.5*	TOGAF* and SAFe 4.5
2022 and Beyond	Private hybrid cloud, containerized, standardized – cloud native apps	Agile and DevOps • SAFe 4.5*	SAFe 4.5*

*In practice, derivatives of these frameworks are used, adapted to the client's situation.

Table 1 — Overview of changing EA and management frameworks over time.

Architecture: Tool to Prevent and Resolve Technical Debt

Another example in combining Agile and EA comes from a company in the consumer sector. This organization had a group of enterprise architects that published guidelines, contributed to the IT strategy, and participated in board reviews. When the company started its first Agile/DevOps program for the rollout of a modern e-commerce platform, the early architectural decisions were left entirely with the technical leads in the development teams. This worked well for the first year, but as the program grew in scope and complexity, the company decided to assign an architect directly to the teams — one architect covering multiple teams — and to create an architecture competence center specifically focused on this program. Across the board, there was more attention to architecture and a standardized, reusable solution.

From a cultural perspective, this proved challenging because the architect's role was unclear. The teams perceived it as a control function, since they now had to justify in detail their decisions and participate in biweekly architecture boards. While the cultural impact still needed to be improved upon, the architect contributed to more communication within the teams, more informed decisions, and a longer-term perspective, asking questions from different angles and bringing into the discussion aspects of which the teams were not previously aware. By working more closely with the development teams, the architect also benefited from the opportunity to observe and get direct feedback on how the architectural guidance and design impacted the execution.

Moreover, with a wider view of the enterprise landscape and direction, the architect was able to identify an architecture backlog and collaborate with the product owner to add and prioritize the relevant items from this architecture backlog into the product backlog. A positive outcome has been better management of technical debt. For example, the e-commerce platform originally had been set up in a data center. While the development team had recognized the need to move to the cloud and identified this as technical debt, the product owner did not give it significant consideration. After the architect joined the team, his input helped the

product owner prioritize this technical debt item and the platform was moved to the cloud.

Streamlining Agile-at-Scale

One of the more successful adopters of Agile, Spotify, has created an organization where teams are allowed to define their own backlogs, select their own priorities, and, in some cases, develop overlapping or concurrent functionalities. A Benelux bank used the Spotify example as best practice in 2015. Using the Spotify model, the bank adopted Scrum as a standard across the organization and became an example for several other non-IT companies on how to implement Agile.

The starting point was the department that developed client-facing applications. The department's personal banking app, developed by the first Agile team, was one of the first and is still one of the most successful applications of its kind. Other departments soon introduced Agile as well, and Agile became the de facto standard across the organization.

There was a downside, though. The adoption of the Spotify model left technology decisions, automation tool choices, use of frameworks, and even sprint durations to each individual team to decide. This resulted in more than two years of effort to rationalize the software catalog by tens of millions of euros and to drive programs and internal promotion campaigns to standardize on a common way of working and an enterprise framework for Agile.

Using this organization as an example, a competing European bank decided to do its Agile implementation slightly different. To roll out Agile across the business units, the bank defined an Agile program supported by architects. Within this program, the bank brought together expertise from different IT partners. Enterprise architecture developed an Agile vision for the organization. The bank involved a consultancy agency to provide Agile coaches who trained internal Agile coaches to take over in the future.

One by one, entire development teams were trained in a common and always improving way of working, based on industry standards. Each team was supported by a solution architect and could reuse all building blocks from a growing architecture repository. The bank regularly captured and analyzed retrospective results and maturity measurements from a growing set of Agile teams and incorporated these in standards and supporting materials. Similarly, it selected and piloted development and testing tools, and distributed those to the Agile teams as well. All these measures greatly increased the adoption rate of Agile.

Architecture as a Servant Leadership Function

Our last example comes from our experience at a US technology company. This organization had a very strong EA function that acted as a checker and gatekeeper. When the company started adopting Agile ways of working, this EA layer got completely disbanded, and architectural decisions were pushed down to the small cross-functional teams to facilitate bottom-up thinking. This proved challenging because it was impossible for any team of eight to 10 people to maintain a comprehensive view of this complex organization. Teams started overlapping in certain areas of responsibility and did not fully understand where they fit in the wider ecosystem. The risks of building functionality that already existed or impacting strategic decisions that the teams were not aware of was too high.

Consequently, this company started rebuilding an architectural function that was very different than the one they previously had. The architecture layer was now primarily solution architects with an enterprise view. T-shaped skills were a must for this new EA group, allowing the architects to be nimble and work closely with business executives, product owners, and development teams alike. Instead of the old architecture review boards, these architects with broad enterprise knowledge and deep technical skills in different areas were now organized informally in a guild.

While decisions continued to be taken by the lowest level visibly impacted, the new architecture function became a trusted advisor, ensuring the decision makers had the knowledge to make informed choices. In this case, the architects did not build an architectural backlog as in our consumer sector example. Instead, they worked very closely with the product owners and the teams to help them prioritize, make decisions, and choose the right funnels to allocate work. Thus, the architects became servant leaders in this organization, playing a central role in creating value by driving integration, enabling the Agile teams, facilitating commitment and consensus, and removing blocks.

Conclusions and Call for Action

Organizations suffering from either or both architecture anti-patterns can benefit from Agile adoption, thereby introducing a faster pace and facilitating more direct communication between the enterprise architects and the Agile teams. The architects must be willing to dive in at the team level, but, in that process, they will become more relevant and valuable to the organization.

On the other hand, organizations "running wild" with Agile and suffering from too much decentralized technical decision making can benefit from architectural thinking, such as business architecture and business prioritization, standardized technical building blocks, nonfunctional cross-team solutions, and backlog prioritization and planning. With solution patterns, standards, and best practices, architects can guide the teams and build a longer-term perspective.

Of course, knowing all this, the goal is to be aware of the value and possibilities that both disciplines have to offer and to implement them simultaneously to prevent pitfalls. It's up to the CIO and the CIO staff to design an IT operating model that combines Agile and EA, considering the maturity level of both Agile and EA in the organization.

Finally, to answer the question in our title: it's neither *agile architecture* nor *architectural agility* — it's both!

Endnotes

[1]Koenig, Andrew. "Patterns and Anti-Patterns." In *The Patterns Handbook: Techniques, Strategies, and Applications*, edited by Linda Rising. SIGS Books, 1998.

[2]"Anti-pattern." Wikipedia (https://en.wikipedia.org/wiki/Anti-pattern).

[3]Zachman, John A. "A Framework for Information Systems Architecture." *IBM Systems Journal*, Vol. 26, No. 3, 1987 (https://pdfs.semanticscholar.org/bda6/aa67d0aaf6ec07d0946244b1563bedc5f861.pdf).

[4]McLeod, Jon. "Enterprise Architecture Is Dead." Medium, 27 June 2017 (https://medium.com/@JonMcLeodEA/enterprise-architecture-is-dead-33dd0e63cbbf).

[5]Roeleven, Sven. "Why Two Thirds of Enterprise Architecture Projects Fail." White paper, Software AG, December 2010 (https://www.cio.com.au/whitepaper/370709/why-two-thirds-of-enterprise-architecture-projects-fail/).

[6]"Part IV: Architecture Content — Architecture Deliverables." *TOGAF 9.1*, The Open Group, 1999-2011 (http://pubs.opengroup.org/architecture/togaf9-doc/m/chap32.html).

[7]Fillié, Jan-Paul, and Hans Boer. "Climb Every Mountain: Overcoming the Barriers to Enterprise Agility." *Cutter Business Technology Journal*, Vol. 30, No. 8, 2017 (https://www.cutter.com/article/climb-every-mountain-overcoming-barriers-enterprise-agility-497036).

[8]Little, Todd. "7 Sins of Scrum and Other Agile Anti-Patterns." *AgileEurope 2016*, Gdańsk, Poland, 30 May–2 June 2016 (https://www.agilealliance.org/resources/sessions/7-sins-of-scrum-and-other-agile-anti-patterns/).

[9]Hughson, Gene. "Accidental Architecture." *Form Follows Function*, 18 July 2014 (https://genehughson.wordpress.com/2014/07/18/accidental-architecture/).

[10]Grundy, John. "Foreword: Architecture vs. Agile: Competition or Co-Operation?" In *Agile Software Architecture: Aligning Agile Processes and Software Architectures*, edited by Muhammad Ali Babar, Alan W. Brown, and Ivan Mistrik. Elsevier, 2013.

[11]"Agile Architecture." Scaled Agile Framework (SAFe) (https://www.scaledagileframework.com/agile-architecture/).

[12]Scaled Agile Framework (https://www.scaledagileframework.com).

[13]"Architectural Runway." Scaled Agile Framework (SAFe) (https://www.scaledagileframework.com/architectural-runway/).

Jan-Willem Sieben is an IT Strategy Consultant at IBM, with a focus on enterprise architecture and the alignment between business and IT strategy. He has been engaged as a consultant, program manager, and enterprise architect in large transformations and innovation programs at central government organizations and a variety of commercial companies, mainly retail. Mr. Sieben has lectured on enterprise architecture for Erasmus University Rotterdam, the Netherlands; University of Amsterdam, the Netherlands; and for the Dutch Association of Enterprise Architects. He can be reached at JWSieben@nl.ibm.com.

Jan-Paul Fillié is a Technology Strategy Consultant, Agile Champion, and TOGAF-Certified Architect at IBM. His focus is on analytics, business intelligence, and data governance, including GDPR (General Data Protection Regulation). Mr. Fillié has worked on IT transformation programs in the financial sector as a consultant, architect, and project manager. As an IBM Agile Champion, he assists teams, programs, and client organizations in Agile/DevOps transformation and implementation. Currently, Mr. Fillié is involved in a global implementation of an enterprise cloud platform as the data functional lead. He can be reached at Jan-Paul.Fillie@nl.ibm.com.

Cristina Popescu is a Business Transformation Consultant at IBM, based in Amsterdam, the Netherlands. She has experience in IT consulting across transportation and logistics, consumer products, industrial, and pharma sectors, as well as in Agile IT operations. Ms. Popescu currently focuses on Agile enablement and business strategy in the consumer products industry. She earned an MSc in accountancy and control from the University of Amsterdam, the Netherlands. She can be reached at cristina.popescu@ibm.com.

FINDING BALANCE

9 Rules of Agile Architecture

by Bob Galen

Software architecture requires balance. Often, you can focus too much on it, creating robust products that miss customer needs or over-engineer solutions. Conversely, especially in Agile contexts, you can under-engineer things and your product efforts can succumb to relentless refactoring rework. So there's a balance to strike in architecture, no matter what methodology you use to create your software. In Agile contexts, that balance is often lost. And it usually leans to less over more.

During the 20 years I've been leading technology organizations to build products, mostly via Agile, I've learned some rules that have helped me — and my teams — successfully strike the right balance. These aren't technically focused rules; they're more generic, so they apply to monolithic, layered, service-oriented, and microservice architectures equally well. Let's dive into the rules and see if you find some value within.

Rule #1: Allow the Architecture to Emerge

At some fundamental level, Agile thinking is experiment-driven. That implies that we want to create prototypes and mock-ups and hack sufficient code to allow us to experiment with different approaches to building our architectures.

We don't want to design in one large lump — ever! Instead, we want to create a layer or certain amount of architecture (services, plumbing, back-end functionality, etc.) and then build something on top of it. That something should be valuable to our client or customer, but not cost too much to build. It should be something easily demonstrated and validated. Something easily changed.

This is what I mean by allowing the architecture to *emerge*. Instead of being presumptuous and building all the plumbing before we layer anything on top of it, we build in slices or increments. There is a term the Scaled Agile Framework (SAFe) community uses called "architectural runway" (sometimes I simply call it "architectural look-ahead"). It is measured by how far your teams are looking ahead to consider architectural implications before building on top of it (or integrating it). Traditional waterfall teams look ahead over the entire project. Agile teams look ahead a few sprints to no more than a release or two.

You should use caution here around the balance between architectural look-ahead and rework. On one hand, if you don't do any look-ahead, then you'll constantly be reworking everything, which will slow you down (or stop you entirely). On the other hand, if you look ahead completely, without experimenting and implementing your ideas, then it will take you a long time to integrate and make it work — again, slowing down or even stopping your progress.

So there is a trick to balancing between looking ahead over the entire project and looking ahead a few sprints (usually relative to the technical and business context you're in). If you're effectively focusing on the journey, then you'll find the right look-ahead balance for your context and your teams.

Rule #2: Treat Your Architecture Like a Product

I've always appreciated it when an organization develops a backlog of architecture stories that it wishes to integrate into its products. The stories are usually different from functional or feature-driven work, in that they might be below the surface or infrastructurally based. But they are important to articulate so they gain visibility. By putting them in a backlog, you start to do things that product owners typically do:

- You groom or refine these stories with the development team(s).
- You define acceptance criteria that capture the essence of what "done" is for each story.
- You discuss the level of effort (points) associated with each story, including testing effort.
- You slice the stories (decomposing them) along execution boundaries.

- You discuss the strategy of how individual stories are implemented to meet an overarching release or architectural goal.
- And, you have value-based discussions, talking about the business value of each story, including the *why* behind each and the customer impact.

An important aspect is the experimentation or exploration part. For example, if you have a feature idea that you think a user would value, you might define a minimum viable product (MVP) for it and whip up a quick/cheap prototype before making a final implementation decision. If the feedback isn't positive, then you'd quickly pivot in another direction.

The point is that I want the same level of thoughtful planning to occur for architecture as for features. In this way, as with features, everyone becomes a stakeholder in the architecture. That means stakeholders understand the motivation, agree with the business case/investment, and understand the customer impact/value of the shared architecture.

Rule #3: A Picture Is Worth …

It may be my narrow experience, but most Agile teams I encounter develop few to no diagrams or high-level views of the architecture they're implementing. Instead, they allude to "being Agile," where architectural documentation is unnecessary, which implies that you simply collaborate around the code and magic (emergent architecture) occurs.

One factor influencing this approach is a fundamental misunderstanding of this Agile Manifesto point: working software over comprehensive documentation. Another factor is that these teams have historically written large-scale documents that have not served them well. Thus, they're scarred by lengthy, but mostly wasted, efforts.

Now, I am an old-school developer who doesn't feel that documentation is inherently bad; particularly high-level, big-picture elements that show teams where they're going from an architectural perspective. A technical roadmap, if you will. Aspects of this architectural roadmap include mapping out a big picture of your business and technical architectural intentions on a whiteboard (virtual, if you can). It is important to align the business view (value streams, strategic roadmaps, high-level personas, story maps, release plans, etc.) with the architectural view (designs, interactions, flows, critical constraints, technical layering, etc.) so that you and your teams have a more balanced view of the goals. This might sound like a lot, but keep in mind that it's a high-level view.

Another aspect of the roadmap is keeping design snippets on a wiki or within your user story definitions. With whatever you document, keep it simple, up to date, and relevant to your teams. The final arbiters of the completeness of your documentation (i.e., whether you've defined enough) are the teams themselves.

Finally, the key to architecture is the same as with the user story: communicating, collaborating, and interacting around the documentation. The conversations are the most important thing. And that means continuous conversations between your business stakeholders, architects, and your teams as well.

Rule #4: Everyone Is an Architect and Everyone Owns the Architecture

There must be a fundamental shift when moving to developing architecture in Agile contexts: from a singular view, where there is an architect that delivers it to the teams for execution, to one where, although there might be experienced architects, *everyone* on the team is responsible for and thinking about sound architecture investments. Instead of it being an individual responsibility, it's an organizational or cross-team responsibility. I often call it the "No Glass House" rule, where we avoid a functional silo (team, group, individual, etc.) solely looking after architecture. Instead, I want *everyone* thinking of solid design, testability, performance, security, user experience (UX) design, simplicity, and maintainability.

In addition to the technical aspects, I also want *everyone* to be thinking of the business rationale behind the architecture. Why are we doing it? What problems are we trying to solve? What's the business case and value proposition? And how does it fit into the flow of the business from a value stream perspective? Some will certainly have more experience in these areas than others, but everyone can put on the hat of the organization, business, and architecture, and consider the implications throughout his or her work.

Another part of this is holding the team (each other) accountable for building wonderfully architected products. The goal should be products that will stand the test of time and wow the users with their intuitiveness, robustness, and reliability.

At iContact, where I was once VP of engineering, we would canvas our customers after moving to Agile to determine what stood out in their minds. Since we were doing quarterly releases (i.e., release train model) and had more than tripled our feature productivity, you would have thought that our speed and increased features would have been top of their minds. It was, but the number one thing that stood out to them was the overall quality improvement of our products and how we were better connecting to their UX needs.

In other words, architecture (quality, robustness, simplicity, etc.) mattered to our customers the most. Those improvements were what grabbed their attention as we continued to evolve and deploy our products.

Rule #5: Keep It Simple and Connect to the Business

This rule is quite near and dear to my heart because I really like complexity. I like engineering complex solutions to simple business and customer problems. And it's also quite comfortable for me to fall into that over-engineering, gold-plating, doing-more-than-is-required mindset.

Why? Because I can. Because I'm an engineer, and the more complex and elegant the solution, the better I feel about my capabilities. It makes me smile.

I think a lot of engineers are like me, but it's the wrong approach; even though it's often easier than thinking deeply about the problem or challenge and then finding the *simplest possible thing* that could satisfy your stakeholders.

One way to combat this tendency is to focus on MVP-like language. Words like "minimum" and "viable" can more narrowly focus our efforts. Another important activity is sitting down with your business partners to understand the *why* behind their requests. People often describe this as a challenge because, as business stakeholders, they are too busy to explain their needs. I'd argue that if they're not too busy to spend corporate funds on solutions, then they shouldn't be too busy to ensure that those solutions meet their needs. Whether they're internal customers, external customers, or both, busyness is often more excuse than it is reality.

If appropriate, including UX activity is an important part of the discovery and design process. Far too often, groups do either too little or too much UX. Either they skip it and dive into solutioning far too soon, or they go into an analysis paralysis state and do UX for months without truly engaging the teams and customers directly.

An important part of business connection is ensuring that stakeholders get into the demos and verify/sign off on the solutions the teams are providing. This notion of demonstration, feedback, discovery, and ultimate acceptance is crucial for closing the delivery loop. It is also vital that stakeholders stick to their word when it comes to those signoffs.

Rule #6: Build in Testability and Resilience

As I defined my rules, I wanted to recognize the quality and testing folks who would be reading them. One of the keys to a solid architecture is considering how your organization will test it. This is true not only from an end-to-end perspective, but also when considering areas like usability, performance, security, reliability, and resilience. You must make these investments in quality and testing transparent to your stakeholders and help them realize their value proposition.

A famous example is Netflix's Chaos Monkey application, which would randomly remove servers and services in its production and testing environments. It simulated various forms of failures, which encouraged teams to improve the resilience and test for it in product development efforts.

In Agile contexts, the focus on test-driven development, behavior-driven development, and acceptance test-driven development also encourages testing, which can be extended to architectural elements. In fact, architecture stories — yes, there can be such things in a backlog — can and should have testability requirements called out in their acceptance criteria.

I consistently try to remind the Agile teams I coach to consider quality and testing in their user story estimates and not to focus solely on implementing the functionality. There's so much more involved in creating robust, testable, and resilient applications. It takes design thinking and, often, support from the underlying architecture.

Don't be afraid to invest in testing automation or infrastructure that eases the burden of and lessens the time for testing. Clearly, Netflix saw the development of Chaos Monkey and similar investments as enabling it

to move more quickly, while also maintaining product resiliency.

Rule #7: Admit That You Don't Know

One of the first things I'd like everyone to acknowledge in software architecture is that we most often haven't done before what we're being asked to do now. That is:

- We're clueless about what the design approach should look like.
- We're clueless about the tools/techniques we'll use.
- We're clueless about the environmental considerations.
- We're clueless about the UX implications and what the customer truly needs.
- We're clueless about the performance implications.
- We're clueless about how to test our solutions.
- And we're clueless about how long it will take to complete our work.

I think the most important acknowledgement or statement that we should all agree to early on in *any* architectural discussion is that *we don't know*. Out of this level of openness and honesty comes the need for prototyping, discovery, and learning. It's hard to do that if we don't look each other in the eyes and say, "We don't know, let's find out."

Once we do that, the focus needs to turn to learning, which is something we can all do. Here are a few techniques for approaching this learning:

1. **Working code is the great leveler.** So, as much as possible, pull together prototype code to learn. The prototypes should be cheap and fast. They're not production code; instead, they're learning code or experimentation code. This includes paper prototypes and similar tools from a UX perspective.
2. **User story spikes are the best way to capture these activities.** Write a spike for each and every major learning activity. Take the time to clarify the acceptance criteria for each. In other words, what key things will we deliver or complete to more fully understand this aspect of the architecture?
3. As mentioned earlier, SAFe has the notion of architectural runway or architectural look-ahead. So, not only do we need to capture spikes and write code, we need to **forecast enough in advance that we're "ahead" of our product development needs.**

Consider working code the ultimate *clarifying view* of your architectural understanding. Aspire to code over study, documentation, and talking about the architecture. Instead, build prototypes as soon as possible, get to working code, and ultimately improve your knowing.

Rule #8: Demo Your Architecture

An extension of the working code points discussed above is demoing your architecture. It is incredible how much pushback I receive on this idea in my coaching. It seems Agile teams are comfortable demoing end-user functionality, but incredibly uncomfortable when you ask them to demo architectural elements. You'll usually hear excuses about there being no UI, or it takes too much extra effort to expose the architecture, or it would be too hard to measure attributes of the architecture in clear business terms. You may also hear that most stakeholders (executives, customers, managers/leaders, etc.) don't really care about their architectural, infrastructural, automation, and other "plumbing-oriented" types of work. They only care about customer-facing features (MVPs) and things they can see, understand, and charge for.

Stakeholders view demonstrations of this sort to be too technical and hard to understand, self-serving exercises that only benefit the teams, or boring and a waste of time. But I like to confront this perception and try to influence stakeholders to endeavor to understand and engage with more technical demonstrations.

Why? Because they *should* care. They are certainly paying for the architecture and they should try and understand the complexity and infrastructural demands within their products. They do not have to understand the architecture the way the teams do, but from the business, value and impact, competitiveness, and investment perspectives, the team's business partners do need to care. Demoing architecture is a wonderful way to provide stakeholders guidance toward this improved understanding. Over time, they'll start to get a feeling for:

- Architectural investment percentages
- Costs associated with their demands/decisions

- Tradeoff decisions
- Risks associated with architecture (implementation *and* delaying updates)
- The drivers behind refactoring
- The investment "mix" of features versus architecture inherent to each of their products

All these elements lead to improved understanding, empathy, and respect for *all aspects* of their products. I've found that stakeholders who embrace their architectural investments are far better decision makers.

Rule #9: Chaos Is Constant, So Continuously Refactor

Let me use an analogy to begin to explain this rule: I am a home owner in the US state of North Carolina. Our climate causes specific types of degradation in my home, and there is simple aging to contend with as well. As a result, I find myself investing annually in the upkeep of my home and its systems. I view it as a "pay me now" versus "pay me later" decision, and I like to keep my home nice, so I lean toward pay me now.

Some of my neighbors have the reverse philosophy. An example can be found in house painting. I've been painting on a regular schedule, every five years. Some of my neighbors paint only when things are obviously falling apart, as evidenced by exposed wood, wood rot, and severely peeling paint.

My strategy is more of a preventive approach and the costs are frequent but low. My neighbors, on the other hand, have less frequent payments when it comes to the exterior of their homes, but when they do pay, it's more costly. For example, they might need to replace all their siding because they haven't painted it in 10-plus years.

I tend to update to new technologies as well. For example, a few years ago, I updated my HVAC systems to much more efficient units. Not only are they more reliable, but I'm saving a lot of money with their increased efficiency.

Switching back to software and architecture, I always recommend the same strategies for software development products and applications. We must acknowledge that software ages *and* our approaches and tools evolve, so we want to continuously invest in the care and feeding of our products. That investment needs to be part of our business case and factored into the ROI.

This isn't just at a feature level. I would argue that it's even more important to keep the plumbing (infrastructure, tooling, automation, architecture, design, integrity, performance, maintenance, etc.) up to date as well. Your stakeholders may not always see this investment, but they will experience whether you are, or are not, making it.

Finally, I am a proponent of asking my teams about technology evolution and trends and determining how we want to invest along evolutionary curves. This includes new technology, new tools, and new approaches. It's important to *listen* to your team and *trust* your team in these recommendations. They'll know far better than you what the relevant trends are and the value that updating can bring to your business and customers.

To return to my analogy, *don't wait until your house crumbles and you must rebuild from the ground up*.

Wrapping Up

That's it. Consider these the nine Rings of Man[1] from *The Lord of the Rings*. Now if I were asked to share the one ring to rule them all, it would be balance: getting to the point where you define, refine, and implement just-enough and just-in-time architecture. May you eventually find that one ring to rule them all.

Endnote

[1]"Rings of Power." Wikipedia (https://en.wikipedia.org/wiki/Rings_of_Power).

Bob Galen is an Agile methodologist, practitioner, and coach based in Cary, North Carolina. He helps guide companies and teams in their pragmatic adoption and organizational shift toward Scrum and other Agile methods. Mr. Galen is currently Director of Agile Practices at Zenergy Technologies and President of RGCG, LLC. He regularly speaks at international conferences and professional groups on a broad range of topics related to Agile software development. Mr. Galen is the author of Agile Reflections, Scrum Product Ownership, *and* Three Pillars of Agile Quality and Testing. *He can be reached at bob@rgalen.com.*

THE LIVING FABRIC

A Light-Touch Architecture Governance Approach

by Miklós Jánoska

Remember the "I see dead people" meme from the movie *The Sixth Sense*? As an architect practitioner, I see dependencies everywhere — sometimes they are empowering, but more often, they are implicit and constraining. Now, I don't mean dependencies visible on the surface (i.e., visible in workflows, impact maps, package dependencies, and intricate UML diagrams). Rather, I am referring to dependencies found in the underlying (and subtle) *living fabric* of the business technology ecosystem (i.e., the implicit architecture, the network of interactions and delivery constraints, and the pulse of the runtime infrastructure); in short, the emergent dependencies.

Over the years, I have found dependency-oriented thinking a powerful tool and dependency awareness an easy common denominator among the different players in the architecture lifecycle. So why can't we build a dependency-based model that is flexible, deep, and broad, and, at the same time, enable architects to answer their most common questions? Thus, I propose a dependency-based, lightweight, pragmatic approach to build architectural insight into the Agile delivery process and continuously reflect the changes inherent in the Agile process.

Despite the multitude of architectural frameworks and methods, experiencing a smoothly working, pragmatic synergy between delivery teams and the architecture discipline is rare. Root-cause analysis brings our focus to semantic gaps (i.e., gaps arising from the continuous erosion of contextual understanding in the development process). While common architecture practices address these via process and control, often such practices produce yet another layer of confusion in the organization. Under the increasing pressure of accelerating marketplaces and rapidly evolving technologies, internal velocity and responsivity become significant differentiating factors. Indeed, responsive, flexible software ecosystems enable high-speed businesses.

This article describes one way of establishing a non-blocking architecture governance practice for Agile development teams. The approach consists of a few independent ideas that lead to organically integrating architecture into the process, the delivery pipeline, and team routines. This article also provides insights into common difficulties in Agile projects — the difficulties of capturing the right level of abstraction, of keeping a pragmatic balance between documentation and the rapidly changing deliverables, and, lastly, of integrating architectural practice into everyday teamwork. In Agile principles, communication and contextual understanding are key, but when it comes to architecture, being aware of tradeoffs and consequences has at least the same level of importance. The emergent approach proposed in this article is easy to introduce (even partially), easy to follow, and easy to adapt to varying team cultures.

A Glimpse into the Agile Architect's Day

Given an established enterprise with its decades-old IT department, processes, and practices versus the accelerating marketplace — *when* missing out on modern IT practices and being too rigid to react to market trends, with even innovation on half-yearly cycles — *then*[1] we see the hiring of a talented Agile architect to bridge the gap and lead the recently established digital pillar of the company.[2]

Let's explore the common challenges The Architect faces via the story of a day.

At the start of the day, The Architect's mind is still full of the escalation emails that arrived last night. Most of them asked about viability and for accurate estimates of important future roadmap items. The rest ranged from a gentle reminder of the yearly roadmap planning cycle through blocker escalations to the postmortem details of a new technology that failed in production. The Architect is determined to prioritize the most pressing blockers first to enable as many teams as possible. A few quick decisions are made — hesitantly — under time pressure and not having enough information. They are guided mostly by gut feeling and instinct. A few responses to escalation emails get sent, asking for more business context in hopes of avoiding the invisible landmines of corporate politics.

Feeling more comfortable, The Architect joins in on early team discussions only to learn that there is no mutual understanding between the implementers and consumers of an API and that another team has no idea of the revenue impact of the long-awaited feature it committed to deliver. Without the right governance, this feels like herding cats, but an over-regulated process would slowly strangle the teams' progress. So The Architect quickly advises the teams to talk, share knowledge, and remember to document it, trying to keep in mind that the architecture metamodel — the tool mandated by the corporate enterprise architecture group — needs to be updated, too. In The Architect's rush to the governance board meeting, reports based on "accurate" figures are quickly put together and some draft slides are produced from memory, not having the real-time insight or tools to produce them on the fly.

Architects spend their days trying to synthesize the old and the new; the static and the dynamic; organic solutions and problems; and, ultimately, dealing with people, process, and architecture equally.

At the architecture governance board meeting, there are fewer people than usual, and The Architect quickly scans those present to sense the changes in power around the table. The Architect knows that important decisions need to be made. But these decisions often reflect the typical uninformed conversation where everything has to be "just right" immediately and are often based on "the single objective truth" that people so blindly believe in. The Architect then quickly sinks into familiar thoughts: how the digital department of the company should design "good enough" flexible solutions instead of producing report after report; and how quality, context, and meaning can be so detached from the measures demanded. Becoming somewhat frustrated, The Architect focuses on the meeting again to quickly note a few important changes in direction, the roadblocks arising from corporate policy changes, and the awkward technology solution that a previously unheard of unit has just forced into production.

It's now time for coffee with the team leads and fellow architects and to convey the most important relevant changes. This is also the appropriate time to gently shape the practice and understanding of the architecture. After a quick argument with the most knowledgeable old-timer — a strong analytical thinker who questions anything not accurate to the most minute detail and who has been reassigned from a monolith company to spread domain knowledge — it becomes apparent that a balance between design and engineering thinking is still far away. More hands-on architecture practice is needed, and a pragmatic evaluation of the next emerging technology stack might be a good candidate.

The Architect's closing thought of the day is the need to catch up with the teams as early as possible the next day, as there is no time for activities such as capturing, rethinking, or redesigning the architecture or solutions. There is only enough time to quickly align the delivery teams in-flight to avoid wasted effort.[3]

Architects spend their days trying to synthesize the old and the new; the static and the dynamic; organic solutions and problems; and, ultimately, dealing with people, process, and architecture equally. Without the necessary contextual insight, usually from more than one context, striking the right balance is extremely difficult, leading to fragile tradeoffs.

It's Not the Trees

"Architecture," for the purposes of this article, is the interconnected structure of relevant people, roles, and visions within a certain context. Defined in this generic way, architecture always exists and is implicitly defined. Consequently, architecture expands through space and time (space being the context for the usual structural representations; time being the context for architecture dynamics). As such, our definition of architecture is far from static diagrams and documentation; rather, it is the living fabric in the business technology ecosystem.

When approaching an architecture representation, a key point is its decomposition into elements, usually leading to a containment-based representation structure. We might then navigate the architecture along the "containing" relations,[4] or using predefined viewpoints. "Navigating along the containing relations" is the ability and constraint that navigation is only possible along hierarchies in the usual drill-down order. By principle, this then constrains what can be seen and in what context.

My proposal is free navigation across layers, taxonomies, and granularity levels. Viewpoints by their very nature define a single context; thus, views might miss an indirect relationship completely. Viewing an organic system only via tree-based representations is akin to the "blind men and the elephant"[5] parable. The multitude of different aspects without a holistic view can be misleading.

To fill the gap, the proposed model should represent transitive relationships and their projections to groups of elements. The model should be navigable in a spherical manner, not limiting the insight to specific taxonomies. Any subset of the architecture should be decoupled from the way it is projected onto views. Forcing the emerging relationships into predefined taxonomies should be avoided.[6]

To reach the ideal balance in governing architecture, the challenge is to harmonize the intentional and the emergent architectures. The former is driven top-down, along tree structures, while the latter relies on loosely structured, bottom-up information flows. Thus, to incorporate both, the core representation model needs to expand accordingly. Moreover, to represent architecturally relevant information, the model should center around structural properties and patterns tagged with additional information about relationships and elements. Cohesion and coupling, the most common structural properties describing component autonomy, need to be explicit. Information about specific elements is naturally reflected in the context defined by their relationships to their neighborhood. In summary, the simplest model is an unconstrained dependency network tagged with metadata on its elements and relationships — the Architecture Knowledge Graph.

Time Is as Important as Space

Similar to the semantics of user stories, which always express change, it is beneficial to describe architecture as a series of changes — structural changes through time. The smaller the gap between the architecture representation of the architecture and its realization, both in expressiveness and timing, the more relevant the architectural changes.

Focusing on Agile organizations, we can consider what might be the natural expression of architectural change through time. With Agile, changes are described through backlog items; therefore, it is a natural extension to define architectural deliverables in terms of epics, stories, and tasks.[7] There are two specific concerns here: (1) the architecture items should use a specific language to support translation to the Architecture Knowledge Graph; and (2) architecture deliverables should precede their respective backlog counterparts.

The idea is to capture architecture epics, stories, and tasks one abstraction level earlier than backlog counterparts. Architecture *epics* are defined during roadmap planning. Architecture *stories* align with the iteration preplanning timeline (e.g., some iterations ahead), and architecture *tasks* are those activities performed on the smallest iteration scale. These descriptions feed into the Architecture Knowledge Graph, expressing either architecture definition or change reflected in the living fabric. While there are many architecture and project management tools based on the same idea, their base model is usually relational.[8]

Ideally, the architecture descriptions follow a simple unified language that can serve as a baseline to feed the architecture dependency model. Of all the architecture description languages, perhaps the simplest one would be a flavor of Gherkin.[9] Custom statements can support the selection of a subgraph (*given*), the intended future context of a subgraph (*when*), and the expected architecture transformation (*then*) on the graph.[10]

DevOps to the Rescue!

While the higher abstraction-level definitions can now be fed into it, the Architecture Knowledge Graph is still missing the bottom-up, emergent information about dependencies. This information becomes contextual in the network of the already-defined, higher-level context from the backlog and architecture design. Looking at the maturity of DevOps practices, plenty of data sources are available. We can easily imagine services, for example, running in a cloud platform surrounded by continuous monitoring, log analysis, and operational insight tools. A few examples include:

- Hypervisor and resource scheduler insights
- Cloud infrastructure metrics via the administrative APIs
- Networking insights available from common monitoring tools
- Application event logs and machine logs on individual instances

On top of the operational insight data, a continuous integration (CI) pipeline provides invaluable information about the static structure of the solutions as well as their behavior under test. This information includes:

- Existing components and their external/internal dependencies
- Existing architecture structure
- Environment configurations
- Static code analysis results
- External assets

> *Assuming the dependencies imposed by the IT strategy and the relevant subset of the existing architecture are captured, we can use the repository during roadmap creation to understand all possible directions and which ones are of high impact.*

We might find it challenging to feed this vast amount of information into the graph. Keeping the original purpose in mind, the continuous stream of low-level metrics is secondary to metric aggregates revealing potential dependencies. That said, nothing prevents the integration of an operational insight tool with the graph while the actual log aggregation and stream analytics workloads are managed separately.

With the intentional and emergent information in place, let's see what questions we can answer with insights provided by the Architecture Knowledge Graph.

Actionable Insight into the Living Fabric

This approach invents no new concepts; it is a natural consequence of considering architecture within its broader context, from the running infrastructure to the roadmap vision. Compared to the broadly available architecture tools, let's review a few differentiators in the Architecture Knowledge Graph:

- Its definition emerged from the jobs to be done, which architecture should fulfill.
- Its design principles follow those of viable, organically evolving systems.
- It doesn't represent architecture as a set of static structures but rather as a continuously evolving system.
- It supports on-demand contextual queries (using a graph query language) without restricting predefined views or aggregates.
- It avoids using the concept of time and instead relies on ordering.
- It explicitly differentiates global- and local-scope computations; it does not try to use a single model for all.

In practice, architecture is better observed as it evolves, as opposed to freezing it and being left with a static structure. Dependencies are not limited to the explicit package or solution dependencies. Dependencies can be enablers and constraints at the same time. For instance, resource availability and networking are constraining dependencies but are also enablers of computation and communication in a running system.

Assuming the dependencies imposed by the IT strategy and the relevant subset of the existing architecture are captured, we can use the repository during roadmap creation to understand all possible directions and which ones are of high impact. Decisions can be easily communicated to the teams via the epics and stories and by interacting with the repository, pulling the relevant aspects. As the team progresses, unintentional dependencies emerge and the ones to be satisfied are checked. When coming to a decision point, the what-if scenarios can be checked based on the existing fabric. Finally, timely reports can be extracted on demand to demonstrate alignment and progress in the delivery of the architecture targets. People in different roles can easily and continuously extract their respective insights. With the ability to identify structural and temporal patterns, best practices can be fine-tuned.

Lightweight Architecture Repository

To summarize, in order to support the high-level scenarios, the core software solution needs to:

- Keep track of architecture definitions (i.e., epics, stories, tasks)

- Keep track of dependencies
- Make definitions and their dependencies searchable
- Make it possible to define cross-layer aggregate views

Thinking about the simplest possible solution, a graph database comes to mind. The global metadata and element search is better served by a specific, freestyle search component leading to the following components (see Figure 1):

- **Dependency database** — graph database (i.e., graph storage)
- **Metadata database** — search engine (i.e., searchable metadata storage)
- **Dependency ingestion API** — multiple APIs, layered by throughput and granularity (i.e., CLI API — verification, feed data, etc.)
- **Definition API** — DSL[11] and import logic (i.e., CLI API — monitoring streaming data, CLI administrator, etc.)
- **Visualization** — integration with any diagramming tool already at hand[12] (i.e., Web API)

Some Implementation Traps

Global aggregates are notoriously difficult to compute on graphs; they should instead be precomputed in a reactive way. As the time dimension translates to ordering relationships in the repository, capturing staged architecture transitions requires extra attention in order to preserve the previous, current, and forthcoming versions of the changing elements. The DSL, used to describe the dependency graph changes, can be easily over-engineered while trying to explicitly represent all low-level detail. Querying graphs might lead to a longer learning curve.

Organizational Adoption

Teams can continuously work on the repository and use it to capture insights; at the same time, the solution is well suited to an internal open source approach. In this way, the architecture team can effectively shape the solution to the given technology stack and existing practices. Thus, the graph organically grows with every piece of discovery and analysis and with every architecture backlog item.

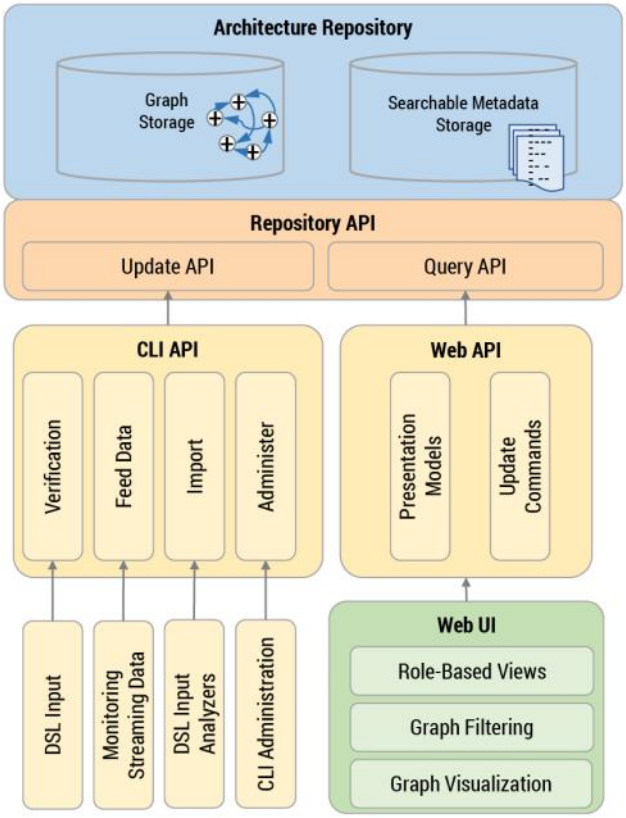


Figure 1 — Schematic architecture.

Emergent Maturity

Without being overly prescriptive about evolving dependency awareness and architecture maturity, there are clear markers of adoption. With increasing contextual insight, teams are empowered to understand the architecture tradeoffs and clearly see the architecture constraints. Controlling the fabric of cross-contextual dependencies, architects can make good-enough, just-in-time decisions. Greater transparency results in higher efficiency in harmonizing the emergent and intentional architectures. The following four aspects help in understanding emergent maturity:

1. **Introspection** — describes how broadly dependency data points are captured and how easy it is to capture them
2. **Emergent architecture** — describes how much emergent information can be accommodated
3. **Dependency awareness and usage** — describes the extent of widespread dependency awareness and how much the repository information covers the overall software development lifecycle
4. **Lightweight governance** — describes how the practice of just-enough intervention, driven by contextual insight, is appreciated

We will now break down each aspect in more detail.

Unlocking new information sources and continuously capturing relevant architecture details and just-in-time interactions with the Architecture Knowledge Graph lead to deeper insight and a contextual understanding.

Introspection

- Ad hoc introspection is typically triggered by bugs/issues.
- We can capture CI and minimal operational insight, test results, package dependencies, and basic runtime metrics.
- We can perform static and runtime code analysis from local to production environments in CI, identify trends via comparisons to historical data, and use dependencies to drive testing on the infrastructure level (e.g., inaccessible service scenario).
- We can collect dependency data points in all environments through the CI and deployment processes; we can partially introduce control points.
- We can capture dependency markers through CI, deployments, and infrastructure; introduce dependency control points in all environments; use dependency management libraries during development; and correlate insights across the graph.

Emergent Architecture

- There is no concept of architecture, or it is considered isolated from the software delivery process.
- Architecture is a set of documents and processes manually updated and loosely aligned with the delivery milestones.
- Architecture repository, centralized view, manual or batch updates, and automated interactions are not fully supported.
- Architecture repository usage is partially automated; repository is navigable across different levels and supports on-demand contextual queries (i.e., freestyle graph queries on the dependency-oriented core model).
- The Architecture Knowledge Graph, updates, and queries are automated and integrated into the lifecycle; on-demand contextual information is available; architecture descriptions and precomputed aggregates are supported.

Dependency Awareness and Usage

- There is no appreciation of the generic concept of dependencies.
- Only particular types of dependencies are observed and handled in isolation (e.g., solution and package dependencies and individual deployments with their resource demand).
- Classes of dependencies are identified and handled partially; the concept of delivery, logical, and infrastructure dependencies emerges.
- Dependencies are classified in alignment with the layers and structure of the architecture; this expands to the delivery and runtime infrastructure areas.

- Dependencies are proactively and intentionally managed and monitored; all architecture layers, the delivery process, and physical realization rely on the dependency graph.

Lightweight Governance

- There's isolated or no architecture governance.
- Governance is based on team collaboration without the support of insights.
- A streamlined governance process is established utilizing a stand-alone central repository; team involvement is based on preliminary orientation meetings.
- A central repository is somewhat integrated, partial insight is available (e.g., batch refresh), and governance controls are manifested in the processes and team best practices.
- A complete feedback cycle through the layers of architecture,[13] delivery, and production systems emerges. Architecture is controlled by the Architecture Knowledge Graph, its integration to the development lifecycle is automated, and dependency insights are continuously fed back.

In conclusion, in an Agile setup, we might just merely follow the flow to integrate delivery with architecture. But unlocking new information sources and continuously capturing relevant architecture details and just-in-time interactions with the Architecture Knowledge Graph lead to deeper insight and a contextual understanding. Contrary to common practice, introducing an architecture repository doesn't have to be big bang; it can be incrementally put together with an enabling base model and some attention to dependencies. Following the improved insight, we can organically establish lightweight governance. Indeed, architecture is alive and continuously changing through space and time; static governance structures freeze it to fragility.

Outlook and Recommendations

While this article does not reference any of the following pointers specifically, they deeply relate to the dynamics and potential future of a balanced architecture governance practice:

- Barry O'Reilly's concept of "architecting for anti-fragility" helps practitioners systematically address fragility in the process and solutions of architecture (see his article, coauthored with Gar Mac Críosta, earlier in this issue of *CBTJ*).
- David Snowden's Cynefin framework gives significant insight into the observability and predictability of simple, complicated, complex, and chaotic systems.[14]
- When considering the impact of observability on architecture, Cutter Consortium Senior Consultant Roger Evernden's reflection about the learning cycle comes to mind.[15]
- Category theory, a branch of mathematics, lays the foundations to grasp the expressive power and beauty of manipulating only nodes and relations.[16]
- For the more demanding, analytical reader, Allen Woods's portal, "The Performance Organisers," provides a journey to structured coherent design, demonstrating linked information-based architecture modeling.[17]
- Nicolas Figay's pragmatic journey into using ArchiMate extended with graph capabilities (aka "enterprise cartography") reveals many opportunities.[18]
- Tom Graves is an infinite source of architecture wisdom, tools, and advice. More specifically, he gives advice on navigating the layers of architecture.[19]
- Randy Shoup's "Minimum Viable Architecture" concept is a real-life buoy when seeking the right balance.[20]
- When thinking about sustainable systems, we cannot ignore Stafford Beer's viable system model.[21]
- Simon Brown's C4 model (context, container, component, code)[22] is a pragmatic method that supports emerging contextual architectures with a rapid learning curve.
- Finally, both Disciplined Agile Delivery[23] from Cutter Senior Consultant Scott Ambler and the Scaled Agile Framework (SAFe)[24] detail Agile architecture in the delivery context.

Endnotes

[1]The "*given-when-then*" structure refers to Gherkin, the common description language of behavior-driven development (BDD); see "Introducing BDD" (https://dannorth.net/introducing-bdd).

[2]This scenario was chosen over another common story: rushing an Agile startup without any established architectural baseline or feedback cycle. In practice, both scenarios reveal similar difficulties.

[3]One might rightly think that time management is the real issue. However, from another perspective, software projects are rarely protected from change and even under careful time management, unforeseen circumstances emerge.

[4]For example, whenever a link is labeled as "part of" or "implements," it is a "containing relation" by nature.

[5]"Blind men and [the] elephant." Wikipedia (https://en.wikipedia.org/wiki/Blind_men_and_an_elephant).

[6]More precisely, tree structures typically imply complete coverage of all elements; there can be no uncategorized ones (this requirement is often addressed by introducing a specific undefined taxonomy item). Our core model relaxes the equivalence class-based model.

[7]This is nothing new; see: "Architectural Runway." Scaled Agile Framework (SAFe) (https://www.scaledagileframework.com/architectural-runway/).

[8]As relational models imply tree structures, they too may be subject to the elephant parable, often leading to the misinterpretation of the contextual information and performance problems.

[9]As mentioned earlier, Gherkin is a scenario description language widely used in the practice of BDD that follows the "*given-when-then*" structure.

[10]The headache that comes with a common architecture description language is that it is defined from a structure definition angle; for our purposes, an architecture manipulation language based on expressing change would be a better fit.

[11]Domain-specific language (might be a customized flavor of Gherkin).

[12]Beyond diagramming, we might enrich the existing architecture tool (e.g., Archi) with the contextual data.

[13]More precisely, this is the "inversion of control" principle applied to the architecture layering and dynamics.

[14]Snowden, David J., and Mary E. Boone. "A Leader's Framework for Decision Making." *Harvard Business Review*, November 2007 (https://hbr.org/2007/11/a-leaders-framework-for-decision-making).

[15]Evernden, Roger. "Sensing — Learning — Exploring — Breaking (SLEB) — A New Model?" Enterprise Transformation Through Enterprise Architecture, 6 April 2018 (http://www.evernden.net/sensing-learning-exploring-breaking-sleb-a-new-model).

[16]"Category Theory." *Stanford Encyclopedia of Philosophy* (https://plato.stanford.edu/entries/category-theory).

[17]Woods, Allen. "The Performance Organisers Product Architecture Overview." The Performance Organisors Ltd., 7 June 2018 (http://www.jitsoftware.co.uk/downloads/prodcat.pdf).

[18]Figay, Nicolas. "Switching from Drawing to Enterprise Navigation Systems with ArchiMate." LinkedIn, 21 July 2017 (https://www.linkedin.com/pulse/switching-from-drawing-enterprise-navigation-systems-archimate-figay).

[19]Graves, Tom. "Linking Enterprise-Architecture with Solution-Architecture." LinkedIn, 10 June 2018 (https://www.linkedin.com/pulse/linking-enterprise-architecture-solution-architecture-tom-graves).

[20]Shoup, Randy. "Evolutionary Architecture: Good Enough is Good Enough" (http://www.randyshoup.com/evolutionary-architecture).

[21]"Viable system model." Wikipedia (https://en.wikipedia.org/wiki/Viable_system_model).

[22]"The C4 Model for Software Architecture: Context, Containers, Components and Code" (https://c4model.com).

[23]"The Disciplined Agile (DA) Framework" (http://www.disciplinedagiledelivery.com/agile-enterprise-architecture).

[24]"Agile Architecture." Scaled Agile Framework (SAFe) (https://www.scaledagileframework.com/agile-architecture).

Miklós Jánoska is Director of Technology Solutions at EPAM Systems, based in Hungary. Building on a programmer mathematician past, his main interest is to find novel solutions to complex problems forging deep, hands-on experience with two decades of broad IT expertise. Mr. Jánoska helps companies reach synergies and resolve tension between the rapidly changing modern industry and the traditionally structured approaches, let it be delivery, architecture, or management. He can be reached at miklos.janoska@mjanoska.com.

CUTTER Business Technology Journal

About Cutter Consortium

Cutter Consortium is a unique, global business technology advisory firm dedicated to helping organizations leverage emerging technologies and the latest business management thinking to achieve competitive advantage and mission success. Through its research, training, executive education, and consulting, Cutter Consortium enables digital transformation.

Cutter Consortium helps clients address the spectrum of challenges technology change brings — from disruption of business models and the sustainable innovation, change management, and leadership a new order demands, to the creation, implementation, and optimization of software and systems that power newly holistic enterprise and business unit strategies.

Cutter Consortium pushes the thinking in the field by fostering debate and collaboration among its global community of thought leaders. Coupled with its famously objective "no ties to vendors" policy, Cutter Consortium's *Access to the Experts* approach delivers cutting-edge, objective information and innovative solutions to its clients worldwide.

For more information, visit www.cutter.com or call us at +1 781 648 8700.